UNIVERSITY OF OSLO
Department of Informatics

A pipeline for high-quality free-viewpoint video

Masteroppgave

Kjetil Endal

July 14, 2011

Contents

List of Figures

Acknowledgments

I would like to than my advisors Alexander Eichhorn and Carsten Griwodz for their guidance and feedback. Thanks to Brendan Johan Lee, Steffan Gullichsen and all the other master students at Simula Research Laboratory for contributing to a good study environment.

I would also like to thank Tomas Kupka, Deepak Dwarakanath and Kristian Haga Karstensen for proof reading my thesis and giving valuable feedback.

A special thanks to my mother and my girlfriend Susanne for being a constant support.

Oslo, July 14. 2011
Kjetil Endal

Chapter 1

Introduction

In traditional video, the viewpoint of the camera is determined at capture time. Free-viewpoint video use multiple cameras to capture the scene. However, the viewpoint is not restricted to the viewpoints of existing cameras, but the user can freely select viewpoints in between cameras where no camera is present. The goal in free-viewpoint video, is to render realistic images from arbitrary viewpoints from captured real-world dynamic scenes.

One of the ultimate goals in computer graphic is to render photo-realistic scenes in real-time, but creating traditional mesh-based models require an extensive amount of work by computer graphics animators, and photo-realistic scenes has proven to be a challenging task to create. Although, as more powerful graphics hardware is produced, it allows more and more complex models to be rendered in real-time. Unlike traditional computer graphics where geometry is the fundamental component, image-based rendering techniques (such as free-viewpoint video) use images as its fundamental components.

1.1 Previous work on image-based rendering

In this section we review the evolution of image-based rendering (IBR), a technique for rendering novel viewpoints using two-dimensional images.

1.1.1 Static scenes

Chen [1] created a virtual environment using panoramic images which allowed the user to rotate and zoom the viewpoint in the scene. The panorama was created by rotating a camera around its own axis and stitching together the captured images. Szeliski and Shum [2] also created a panorama by stitching together images, but the images were captured using a hand-held camera. However, the viewpoint in these methods can not be moved. In McMillan and Bishop's Plenoptic modeling [3], the viewpoint could be moved by capturing multiple panoramic images.

Light field rendering [4] and the lumigraph [5] allows creating novel viewpoint by using a dense sampling of light rays. Light field rendering generates a new image of a scene by appropriately filtering and interpolating a pre-acquired set of samples. The lumigraph is similar to light field rendering, but it applies approximated geometry in order to improve rendering performance. However, both these methods require a vast amount of images (hundreds or even thousands).

Chen and William's view interpolation [6] creates novel viewpoints given two images and dense optical flow between them. However, they require pre-rendered images to construct the optical flow field, and does not use real imagery. View morphing [7] extends image morphing techniques to create novel viewpoint between images. Although some correspondences between the views can be calculated automatically, it still relies on user interaction.

When dense depth maps are provided for an image, 3D warping [8] can be used. Novel viewpoints are created by projecting the pixels in the original image to its 3D location, and subsequently reprojecting them onto the novel viewpoint. This is the method applied for our rendering system, and in depth explanation can be found in chapter 5.

Layered Depth Images [9] store depth information not only for the visible surface in an image, but also what lies behind it. Each pixel in the input image contains a list of depth and color values where the ray intersects with the environment.

Debevec [10] created computer models of architecture, although some amount of user interaction is required. The model is texture mapped using the different cameras, and view synthesis can be performed from any viewpoint.

1.1.2 Dynamic scenes

All the aforementioned methods only handle static scenes. Video-based rendering (VBR) extends IBR to also handle dynamic (i.e. time varying) scenes. Virtualized Reality [11] was one of the first VBR systems. Their system incorporated 51 cameras placed around a 5-meter hemisphere looking inwards. The acquisition system first captures the scene using synchronized cameras, followed by an off-line digitization step. Depth is then reconstructed from multiple cameras using a multi-camera stereo method. Viewpoints can then be created from arbitrary positions by creating a textured mesh composited from depth data and multiple images. A similar system was constructed with Immersive Video [12] using 3-6 cameras. However, their system required the geometry of the static background to be modeled beforehand. This prevents the dynamic objects in the scene to occlude the background, since background information is already known.

A different approach use the image-based visual hull [13] as an approximate geometric description of a 3D scene. A visual hull is created by first detecting the silhouettes of foreground objects in a collection of images. The silhouettes are then cast to 3D space and the cast volumes are intersected, which gives the objects visual hull.

The Stanford Light Field Camera [14] consisted as a 6-camera system, later extended to an array of 128 cameras [15].

Carranza et al. [16] used marker-less motion capture to fit the captured movements of an actor onto a human model, which was textured with color information from the cameras. The textured model could then by viewed from arbitrary viewpoints.

Zitnick et al. [17] used a sparse set of cameras arranged in an arch to capture a scene. Novel viewpoints are generated by estimating depth for each camera, and warping the image to the desired location. Although this methods yields high-quality viewpoints, depth estimation is performed as an off-line process.

1.2 The free-viewpoint pipeline

Our free-viewpoint pipeline (see figure 1.1) is divided into three modules, a camera calibration module, a depth estimation module and a free-viewpoint rendering module. First camera parameters are estimated as an off-line process. Both internal parameters

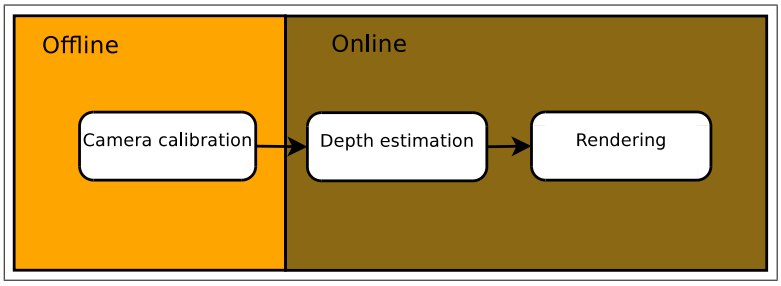


Figure 1.1: The free-viewpoint pipeline

of the camera, and the cameras rotation and position with regard to each other are estimated. Camera calibration is performed once, prior to capturing the scene, assuming the internal parameters are not changed and the cameras are not moved in relation to each other. After calibration, continuous capture of the scene is performed using two cameras, estimating depth and rendering a novel viewpoint for each captured pair of frames. Using the estimated calibration parameters, the pair of captured images are transformed so that they lie in a common plane, a process known as image rectification. Disparity between the images, which is the horizontal shift of corresponding pixels, are then estimated giving us the disparity map. The disparities are then converted to depth values by triangulation. Novel viewpoints are then rendered by back-projecting the image pixels to 3D space (using depth and calibration data) and thereby reprojecting the pixels onto the virtual camera's image plane.

1.3 Applications

Free-viewpoint video has many applications, among others in entertainment and communication.

1.3.1 Tele-presence

Tele-presence applications attempts to create the feeling that the participants are physically present next to each other. Although, the participants in tele-presence and video chat applications are in general not able to have eye contact. This is because a participants must look into the display to see the other participant, and not into the cam-

era which is located outside of the screen. This breaks the illusion of being physically present with the other participant, since a regular conversation typically involves looking into the eyes of whom you are talking to. Free-viewpoint video can be used to create a virtual camera in front of the display, thus allowing both participants to have eye contact.

1.3.2 Multi-view displays

Recently, stereoscopic displays have become commercially available for the consumer market, allowing the viewer to watch 3D video content. The 3D sensation varies greatly with the amount of parallax between the images, although it can cause eye-strain if the parallax is too large. Using free-viewpoint video, synthetic images can be generated at arbitrary levels of parallax, allowing the viewer to decide between the amount of 3D sensation and viewing comfort.

Multi-view displays can show a different image, depending on the viewing angle, much like a hologram. The numbers of cameras necessary for capturing images for multi-view display can be significantly reduced by using free-viewpoint video since a chain of viewpoints between two cameras can be generated.

1.3.3 Free-viewpoint video

Free-viewpoint video, in its most general form, allows for interactive view selection. In traditional video, we are limited to the view selected at capture time. But in free-viewpoint video the user can select any viewpoint, e.g. the user might wish to change the viewpoint in football matches, to get a better viewing angle to determine if a player was offside, or the user can change the viewpoint in training videos to get a better look at what the instructor is doing from any angle.

1.3.4 Special effects

Free-viewpoint video also has many application to special effects. In the movie "The Matrix" a "bullet-time" effect (i.e. freeze-frame) was used, where the video rotates around a scene where time is frozen or in slow-motion. In that setup however, a large

number of cameras were used. Free-viewpoint video allows the amount of necessary cameras to be reduced.

By using recovered depth information from the scene, objects (real or virtual) can be inserted or removed. In the movie "Benjamin Button", a face was first captured and then generated into a 3D model and finally inserted onto the face of an actor. Recently, a similar technique was used in the computer game "L.A. noire", where the heads of actors where captured and turned into a 3D model. The reconstructed heads and facial motions were later inserted onto the computer animated bodies of characters in the game.

1.4 Problem Statement

In this thesis we investigate and implement the components of a real-time free-viewpoint video pipeline. An approach that has shown large promise for high-quality free-viewpoint video is based on camera calibration, depth estimation and 3D warping [17]. We want to find out how the different components and processing steps of such a system affects quality and how quality can objectively be determined. In doing so, we get a better understanding of how quality can be improved and measured. Although a real-time system is desired, we focus mainly on quality.

For our investigation we implement a complete system for free-viewpoint video, namely calibration, depth estimation and free-viewpoint rendering. We then experimentally test our system to find out how the different parts perform with regard to quality and processing time. Our system is not comprised of a single pipeline, but rather from multiple methods for depth estimation and rendering.

1.5 Main Contributions

In this thesis we investigate a pipeline for high-quality free-viewpoint video. Our contributions are threefold; 1) we implement the modules comprised in a free-viewpoint video pipeline, namely camera calibration, depth estimation and free-viewpoint rendering. 2) We analyze the quality and robustness of the implemented modules, do performance measurements using objective quality metrics and assess the implemen-

tation's suitedness for achieving real-time performance. 3) We present variations of the rendering algorithms to improve quality of the rendered viewpoints, including a novel cross-checking constraint that removes artefacts while preserving "correct" pixels in the rendered view.

Implementation We have implemented a pipeline for high-quality free-viewpoint video based on strongly calibrated cameras (i.e. the position, orientation and internal parameters of the cameras are known), depth estimation and free-viewpoint rendering using modified 3D warping techniques. In this thesis, these three modules are subsystems in our free-viewpoint pipeline. However, camera calibration, depth estimation and free-viewpoint rendering also has other application areas and the modules can also be used in contexts different from our scenario.

We implement a calibration system, to estimate the position, orientation and the internal parameters of the cameras. A depth estimation module is implemented using five different disparity estimation algorithms from the Open Computer Vision library (OpenCV) [18]. In the current implementation of OpenCV (i.e. v2.2), only the left disparity map is given as output. We present a simple solution to how the right disparity map also can be estimated, by flipping the input images around the vertical axis and switching the place of the images in the input parameters. Additionally, we show how disparity can be estimated over the whole image in OpenCV, by padding the borders of the input images.

We have implemented three different free-viewpoint rendering algorithms from literature. Variations of the three base implementations are also implemented by taking the best elements from those algorithms, and other improvements given in the literature.

Suggested improvements We present several variations to the free-viewpoint rendering algorithms in order to improve on the quality of the rendered viewpoints. The variations consist of taking the best parts of the implemented rendering algorithms, and also adopting other methods presented in the literature. In addition, we present a novel constraint to improve the quality of the rendered view. Inaccuracies in depth estimation and the blending of foreground and background pixels in image capture cause artefacts, which appear as a "ghost" or corona around the foreground objects.

These pixels can be removed, however many methods also remove non-artefacts pixels as a side-effect of this process. We present a novel constraint that removes "ghosting" artefacts without removing good pixels.

Performance analysis We show that our calibration robustness can be increased by using a large set of images of the calibration object.

The five different disparity algorithms are assessed to determine which are suited for use in a free-viewpoint video setting. The disparity estimation algorithms are tested using standard test data containing ground truth disparity. Computational speed is also measured to find out which algorithms are capable of real-time performance.

The free-viewpoint rendering algorithms are assess using objective quality metrics and a thorough subjective analysis of artefacts in the rendered viewpoints. Quality measurements are performed on both the final rendered image and on sub-steps in the different rendering algorithms. We then compare overall quality of the three implementations and of the sub-steps in order to find out how quality can be improved. Objective and subjective quality assessment is also performed on the different variations of the implemented algorithms, showing that several of the suggested improvements give a better objective and subjective quality.

1.6 Outline

In chapter 2 we introduce the projective and epipolar geometries and the pinhole camera model, which is much of the mathematical foundation used in this thesis. Then, in chapter 3 we explain the process of camera calibration. We explain how accurate calibration can be achieved and present our implementation and our experimental results using our setup. In chapter 4 we explain how depth can be estimated from 2D video content from multiple cameras, and how high-quality depth can be acquired. At the end of the chapter we present our experimental results using the OpenCV disparity estimation algorithms on both standard test sequences and one which we captured with our setup. In chapter 5 we explain 3D warping, a method for generating novel viewpoints using calibration data and the estimated depth. We then investigate the artefacts produced by this simple algorithm and show how high-quality synthetic viewpoints

can be generated. Our three free-viewpoint rendering implementations, based on literature, are then presented, followed by variations of these implementations. The chapter ends by presenting our experimental results for the rendering implementations, a comparison between the different steps in the algorithms and how the suggested improvements perform. Finally a discussion of the results, our concluding remarks and future work are presented in chapter 6.

Chapter 2

Projective geometry and the pinhole camera model

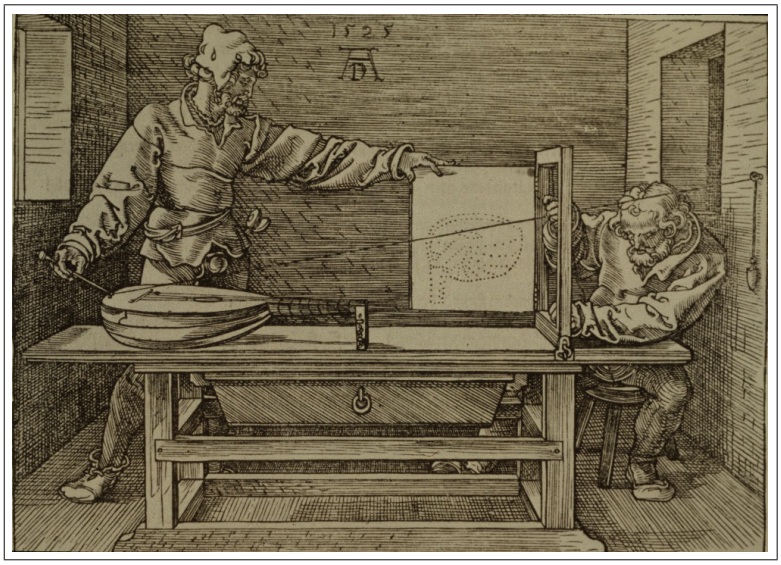


Figure 2.1: Man Drawing a Lute (The Draughtsman of the Lute), woodcut 1525, Albrecht Dürer.[2]

[2] `http://de.wikipedia.org/wiki/Raytracing` (last accessed 2011-07-14)

2.1 Introduction

Albrecht Dürer's woodcut "The Draughtsman of the Lute", shown in figure 2.1, illustrates a projective mapping from a real three-dimensional objects to a planar surface. The lute is projected onto the drawing by making a line from the lute to a single point on the wall, the center of projection (COP). The string intersects with the drawing in the projective plane. In this chapter we present some of the essential mathematical background for this thesis. Specifically, we introduce the pinhole camera model, which explains how real world objects are mapped onto an image plane. Unfortunately this simple ideal model does not sufficiently model real-world cameras. We therefore proceed by explaining how properties of real cameras, such as distortions caused by the lens, affect the image formation process and how these distortions can be removed so that the pinhole camera model holds. We then explain the geometric relationship between two cameras, i.e. the rotation and translation between them. The pinhole camera models is then reformulated using homogeneous notation to simplify the mathematics. A brief introduction to epipolar geometry is then given, which describes the relationship between the image planes of two cameras. At last we explain a method called rectification that transforms two views into row aligned coplanar views.

2.2 The pinhole camera model

Consider a box which is light-proof except for a tiny hole (the aperture) in one of its sides. Rays of light pass through the aperture and an image is formed inside the box on the opposite side of the aperture. Since the aperture is so small, only a single ray of light from any given point in the scene can enter the box, which causes the image to be inverted. This contraption is known as a pinhole camera as illustrated in figure 2.2.

In this ideal pinhole camera, the size of an image is related to the real three-dimensional object by the focal length, which is the length between the aperture and the image plane. In figure 2.3 we have the focal length f, the height of an object X in space and the height of the same object x projected on the image plane. By similar triangles we see that

[3] `http://www.acmi.net.au/eli_iv.htm` (last accessed 2011-07-14)

Figure 2.2: A pinhole camera.[3]

$$\frac{-x}{f} = \frac{X}{Z} \tag{2.1}$$

, where Z is the distance from the pinhole plane to X (i.e. the object) along the optical axis. If we solve equation 2.1 for x we get

$$x = -f\frac{X}{Z} \tag{2.2}$$

Equation 2.2 is often reformulated by moving the image plane in front of the camera as shown in figure 2.4, so that the image is no longer inverted. The COP is the point where all the rays of light intersects, located at the pinhole. The line going through the COP perpendicular to the image plane is called the *optical axis*, the point where it intersects the image plane is known as the *principal point* and the plane which goes through the COP parallel to the image plane is known as the *principle plane*.[4]

A ray of light from a point in three-dimensional space $Q = (X, Y, Z)^T$ passing through the COP intersects the image plane at a point $p = (u, v)^T$ where u and v are at

[4]The COP is also known as the camera center or the optical center, the optical axis is also known as the principal axis or the principal ray and the principal plane is also known as the pinhole plane.

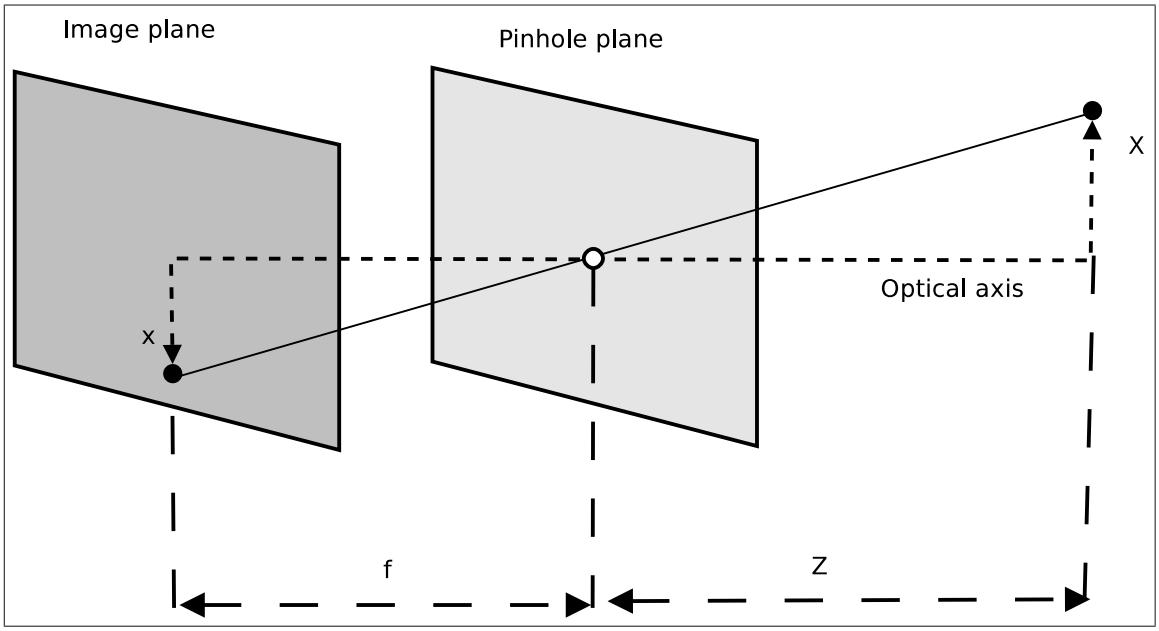


Figure 2.3: Pinhole camera model.

integer pixel positions. We then have:

$$u = f\frac{X}{Z} \text{ and } v = f\frac{Y}{Z} \tag{2.3}$$

2.2.1 Intrinsic camera parameters

The intrinsic camera parameters model the internal parameters of the pinhole camera model and explain the geometrical relationship between the pinhole and the image plane, physical properties of the pixels on the image sensor and distortions caused by the lens in real cameras.

Focal length

The focal length is the distance from the center of the lens to the image plane along the optical axis.

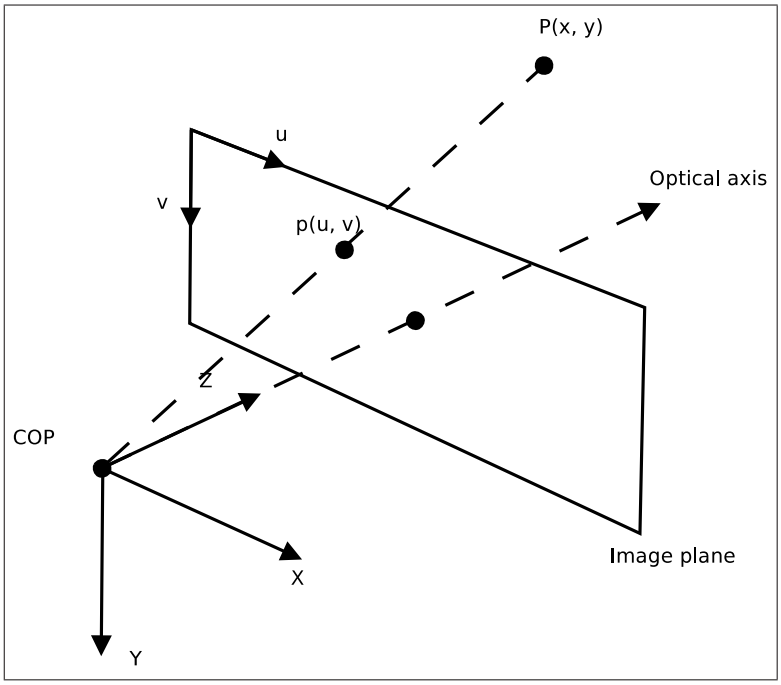


Figure 2.4: Pinhole camera model where the image plane is moved in front of the focal point.

Principal point offset

Intuitively one might think that the principal point is located at the center of the image. This is rarely the case in real cameras, since that would require the image sensor to be perfectly positioned in the manufacturing process of the camera. In addition, it is common to place the origin of the image coordinate system at the top left corner in image/video processing systems. We rewrite equation 2.3 with an offset c_x and c_y along the x-axis and the y-axis respectively, giving us

$$u = f\frac{X}{Z} + c_x \text{ and } v = f\frac{Y}{Z} + c_y \tag{2.4}$$

Pixel aspect ratio

The pixel aspect ratio is the ratio of the width and height of a physical pixel on an image sensor. Most image sensors have square pixels, although some system use rectangular pixels. To account for this the focal length is scaled with the width and the height of

the pixel. We then get

$$u = s_x \cdot f\frac{X}{Z} + c_x \text{ and } v = s_y \cdot f\frac{Y}{Z} + c_y \tag{2.5}$$

where s_x and s_y is the width and the height of a single pixel.

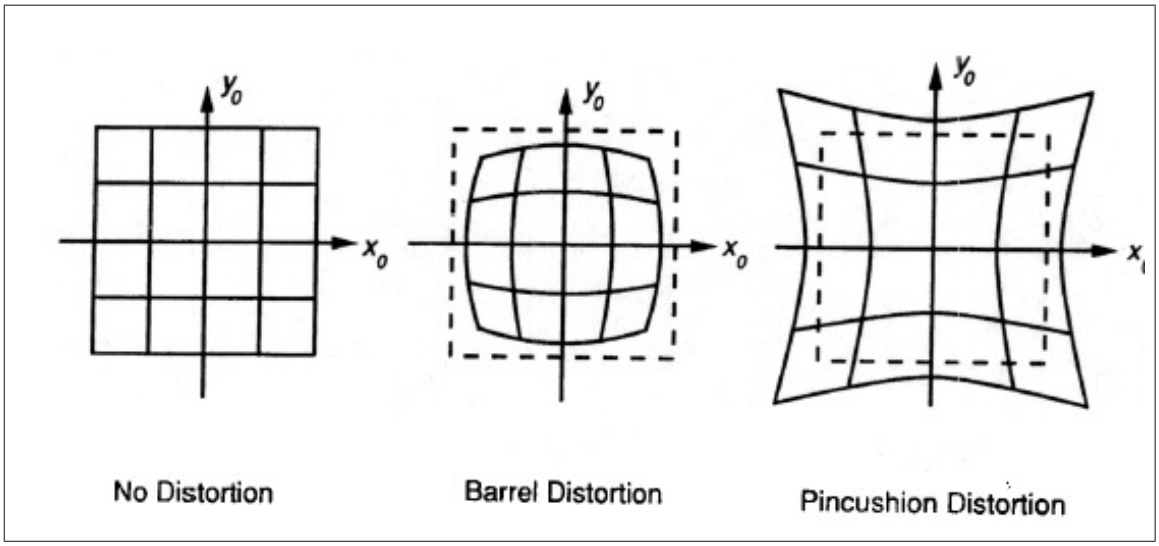


Figure 2.5: Radial distortion. Undistorted image (left), barrel distortion (center), pincushion distortion (right).[6]

Lens distortion

A real pinhole camera is impractical to use since it does not gather enough light for a fast exposure time. Real cameras have much larger apertures than pinhole cameras, to let more light through. To prevent the image from being blurred and to allow rapid exposure time lenses are used to gather and focus the light onto the image sensor. Unfortunately, by introducing lenses we move away from the simple geometric model of the pinhole camera. The use of a lens adds distortions to the image, in particular radial lens distortion and tangential lens distortions[7]. Complex systems of lenses are

[6] `http://www.uni-koeln.de/~al001/radcor_files/hs100.htm` (last accessed 2011-07-14)

[7] There exists a number of other distortions caused by the lens, though radial and tangential distortion are usually the most prominent.

sometimes used (in more expensive cameras) to minimize the distortion, but any real camera system has distortions from the use of lenses.

Radial lens distortion comes from the spherical shape of the lens which causes the light to be bent. The distortions extends outwards from the center of the lens (hence the name radial). In practice, this makes straight lines in the real world to be mapped as curved lines in the image (see figure 2.5. However, we can correct this distortions so that the pinhole camera model still remain valid. The corrected coordinates of each pixel is then given by:

$$u_u = u_d(1 + k_1 r^2 + k_2 r^4) \tag{2.6}$$

and

$$v_u = v_d(1 + k_1 r^2 + k_2 r^4) \tag{2.7}$$

where u_u and v_u are the undistorted pixel positions, u_d and v_d are the distorted pixel positions in the x and y direction respectively, k_1 and k_2 are the radial distortion coefficients and r^2 is the normalized radial distance from the principal point.

Tangential lens distortion appears when the lens is not parallel to the image sensor (see figure 2.6. Correcting for tangential distortion we get:

$$u_u = u_d + 2p_1 v_d + p_2(r^2 + 2u_d^2) \tag{2.8}$$

$$v_u = v_d + p_1(r^2 + 2v_d^2) + 2p_2 u_d \tag{2.9}$$

where p_1 and p_2 are the tangential distortion coefficients.

2.2.2 Extrinsic camera parameters

The extrinsic camera parameters are the external parameters of the cameras (i.e. orientation and position), typically given as a 3-by-3 rotation matrix and a 3-dimensional translation vector in the world coordinate system.

[8] `http://www.flickr.com/photos/riseriyo/4558440101/` (last accessed 2011-07-14)

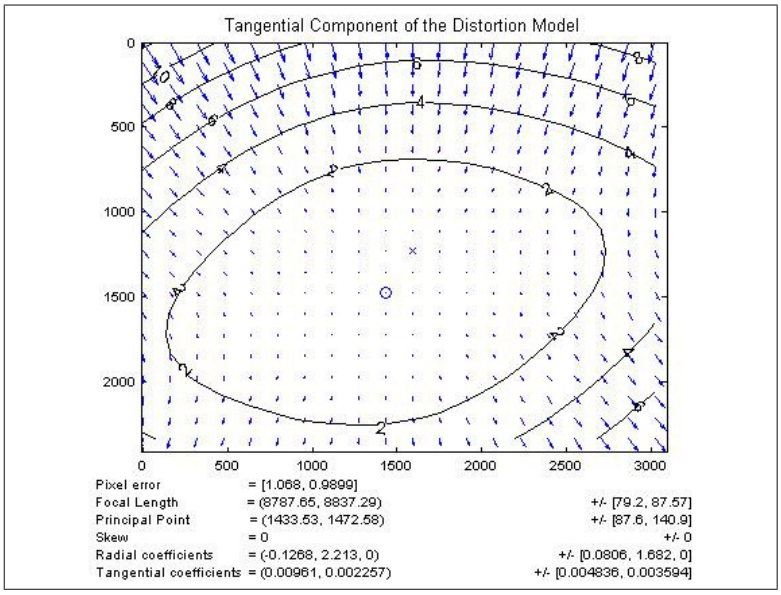


Figure 2.6: Tangential distortion. [8]

2.2.3 Perspective projections using homogeneous coordinates

In projective geometry, unlike Euclidean geometry, points at infinity are treated as traditional points. These "extra" points at infinity can easily be described using homogeneous coordinates. In Euclidean space a point can be described by inhomogeneous coordinates and is represented by a 3-dimensional vector $(X_i, Y_i, Z_i)^T$. In the projective space, the point is described using a 4-dimensional vector (X_h, Y_h, Z_h, λ), where λ is a free scaling parameter called the homogeneous scaling factor. Cartesian coordinates can be converted into projective space by using:

$$X_i = \frac{X_h}{\lambda}, Y_i = \frac{Y_h}{\lambda}, Z_i = \frac{Z_h}{\lambda} \tag{2.10}$$

where $\lambda \neq 0$. The use of homogeneous coordinates simplifies the perspective projection, as it can now be solved using a system of linear equations.

We now reformulate the pinhole camera model in homogeneous coordinates. Let $Q_w = (X_w, Y_w, Z_w, 1)^T$ be a point in space and $p = (u, v, 1)^T$ its projected point in the camera's image plane in homogeneous coordinates. The relationship between Q_w and p is given by $\lambda p = PQ_w$, where P is the projection matrix $P = K \begin{bmatrix} R & t \end{bmatrix}$, K is the

intrinsic matrix represented by internal camera parameters, where f denotes the focal length, c_x the pixel width, c_y the pixel height and $\begin{pmatrix} o_x & o_y \end{pmatrix}^T$ is the principal point.

$$K = \begin{bmatrix} fc_x & 0 & o_x \\ 0 & fc_y & o_y \\ 0 & 0 & 0 \end{bmatrix} \tag{2.11}$$

We can then project a point from camera to world coordinates by

$$\lambda Q_w = R^{-1} \cdot K^{-1} p \cdot d(p) - R^{-1} \cdot t \tag{2.12}$$

where p is the pixel coordinates, $d(p)$ is the depth at pixel coordinate p and t is the translation vector.

2.3 Epipolar geometry

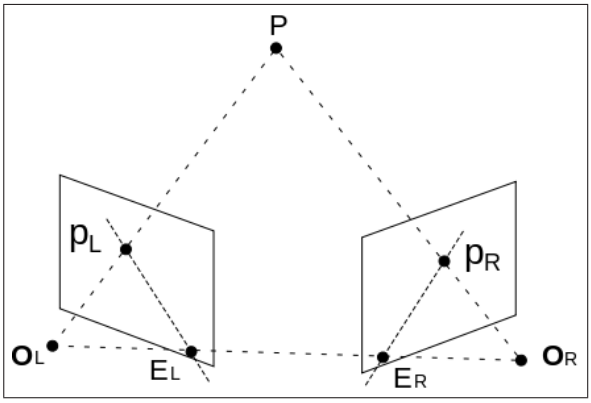


Figure 2.7: Epipolar geometry. [9]

Epipolar geometry is the geometry between two views, described by the intrinsic parameters and the views relative position and rotation. Consider a point P in space projected through the centers of projection O_L and O_R onto the image planes at position p_L and p_R as shown in figure 2.7. The centers of projection and the point P uniquely determine a plane known as the epipolar plane π. Given the point p_L, the corresponding

[9] `http://en.wikipedia.org/wiki/Epipolar_geometry` (last accessed 2011-07-14)

point p_R must lie on the epipolar line $e_R - p_R$, this is known as the epipolar constraint. The points e_L and e_R are the epipoles, i.e. the intersections of the line between the centers of projection O_L and O_R and the image planes.

Since the points p_L and p_R lies on the same plane we have a constraint for where we should search for point correspondences. Having a point p_L (which is the projection of P onto the image plane) we know that p_R must lie where the epipolar plane π intersects the image plane. This line where the epipolar plane intersects the image plane is known as the epipolar line.

- Epipolar plane - The plane which is determined by a three-dimensional point and the two camera centers.
- Epipolar line - The line where the epipolar plane intersects the image plane.
- Baseline - The line between the camera centers.
- Epipole - The point where the baseline intersects the image plane. This is also the same as the projection of one camera center onto the other image plane.

2.3.1 Rectification

In a frontal parallel camera configuration the cameras have identical internal parameters and the optical axes of the cameras are collinear. In this configuration epipolar lines are horizontal and parallel. However it is difficult to orient and align the cameras in practice. Instead of manually aligning the cameras we can transform the images onto a common synthetic image plane (i.e. the images are coplanar) so that the epipolar lines are horizontal and parallel, we refer to this transformation as image rectification. This can be seen as rotating two virtual cameras so that they align, and adjusting the internal parameters so that the images have the same size as illustrated in figure 2.8. Image rectification is done to simplify the search for corresponding pixels along horizontal lines, instead of searching along the epipolar lines.

[11] `http://en.wikipedia.org/wiki/Image_rectification` (last accessed 2011-07-14)

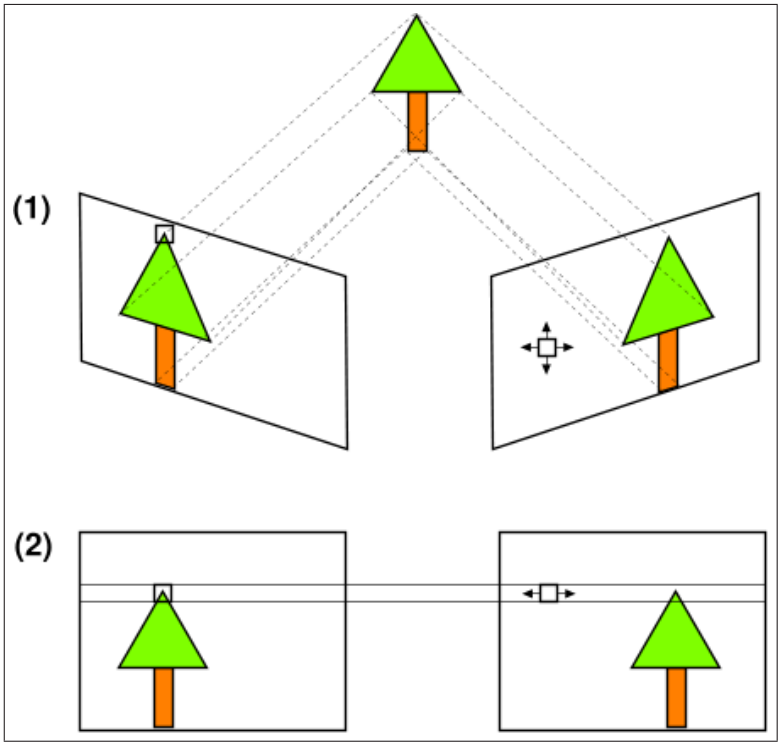
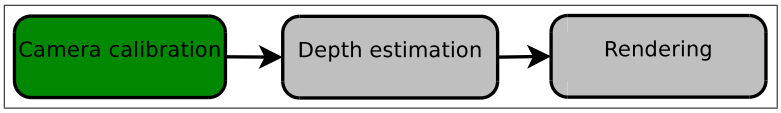


Figure 2.8: The pair of images in (1) are rectified as shown in (2), so that corresponding points lie on a horizontal line in the two images.[11]

Chapter 3

Camera calibration



3.1 Introduction

In this chapter we explain the process of camera calibration. We first introduce the concept of camera calibration and how accurate calibration can be performed in section 3.2. In section 3.3 we present our implementation of a camera calibration system. We then explain how camera calibration can be assessed in section 3.4, and conclude the chapter by presenting our calibration results in section 3.5.

3.2 Camera calibration

Camera calibration is a process of estimating the intrinsic and extrinsic parameters of the camera. Calibration is necessary in order to relate image coordinates and real world points, e.g. using perspective projections (see section 2.2.3). It further allows us to utilize the epipolar constraint by pair-wise rectifying images prior to depth estimation, which significantly simplifies the correspondence problem presented in section 4.2. It

is clear, that in order to apply the pinhole camera model, intrinsic, extrinsic and distortion parameters should be accurately estimated. Failing to do so reduce the quality of both depth estimation and rendering. The calibration process determines the relationship between 3D points of known geometry and their projections on the 2D image plane by calculating the parameters that minimizes the distance between the measured 2D points in the image plane and their corresponding 3D points by applying the calibrated camera model. Calibration relies on both precise knowledge of the calibration object and accurate detection of the feature points in the image plane [19]. The calibration quality is usually a trade-off between the required accuracy and the manufacturing complexity and cost of the calibration rig. A checkerboard is frequently used due to its simplicity in manufacturing (it can be printed on a regular printer), accurate detection of feature points at sub-pixel positions [20] and robustness to radial distortion [21]. Although inaccurate printing of the checkerboard can be a source of error [19]. A flexible and accurate technique from Zhang [22] estimates the intrinsic, extrinsic, radial and tangential distortion parameters using a planar calibration grid. Due to the planar surface of the calibration object, at least two views from different position are required, although more views are usually sought for robustness. However, poor planning of the captured viewpoints can lead to degenerate configurations where accurate calibration can not be achieved. Degenerate configurations occur when additional images of the checkerboard does not add additional constraints on the calibration, such as when the checkerboard undergoes a pure translation [22]. The calibration pattern (or the camera) should therefore be rotated as well. In addition there exist a strong projective coupling between the principal point and the tangential distortion [23] and the intrinsic and extrinsic parameters in general [24]. It is also reported that small angle variation between the views used for calibration can lead to a degenerate configurations [22]. Therefore a set of different views, with a high variation of orientations should be used in order to get an accurate estimation of the camera model. Zhang [22] reports that angles of 45 degrees (around an arbitrary axis) seems to give the best performance and that a larger number of views lead to a more robust calibration.

3.3 Implementation

To calibrate the cameras we estimate both the intrinsic and extrinsic parameters by using a calibration rig of known geometry. We take images of multiple perspectives of a checkerboard pattern and estimate the location and orientation with regard to the world coordinates and the internal parameters of the camera. The cameras are calibrated in two steps, single camera calibration and stereo camera calibration using OpenCV.

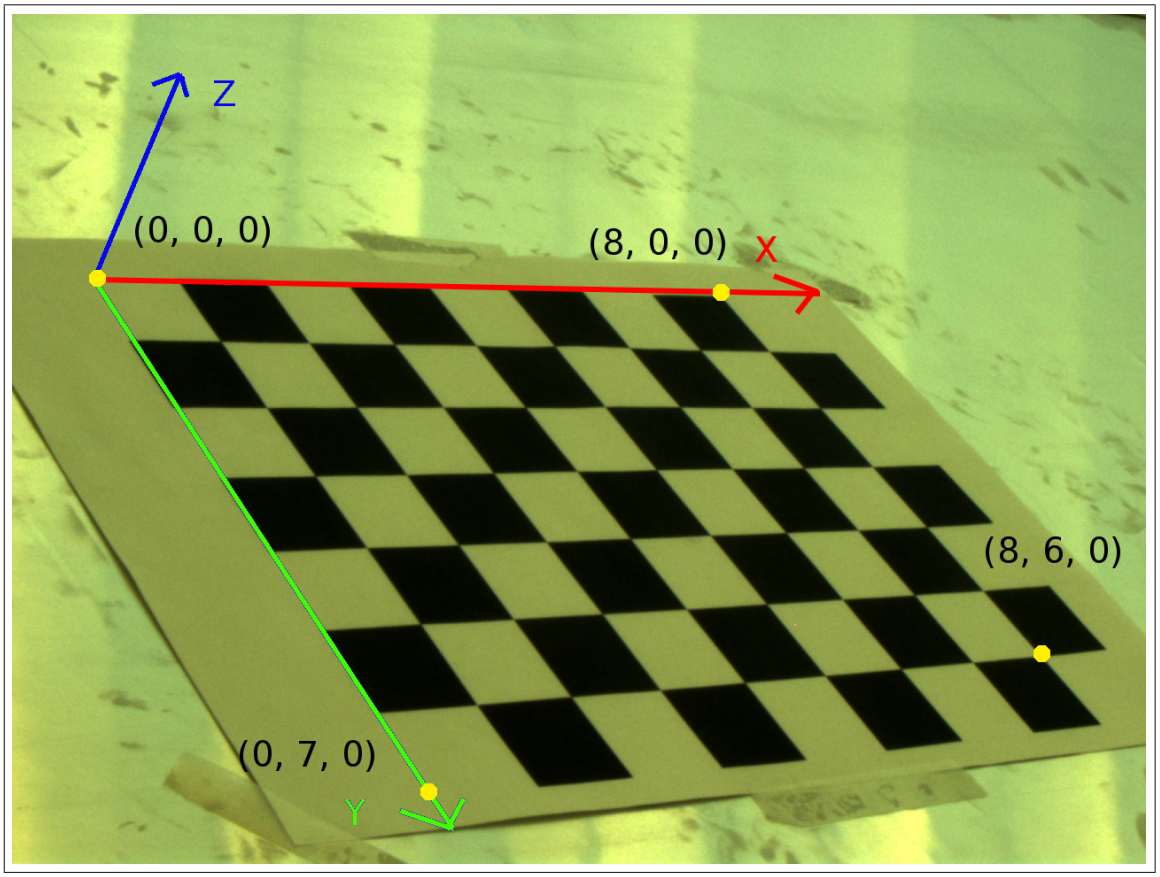


Figure 3.1: The checkerboard's coordinate system. The calibration is carried out by finding the relationship with the object points (the corners in the the checkerboard's coordinate system) to the image points (the detected corners in the image).

Single camera (intrinsic) calibration In the single camera calibration we estimate the intrinsic and distortion parameters for each camera separately. Prior to the calibration we have printed out a black and white 9 by 7 checkerboard pattern on a laser printer. Using the checkerboard and a single camera, the calibration procedure is performed by:

1. Capturing $N \geq 2$ images of the checkerboard pattern with varying angles.
2. Identifying the inner corners of the checkerboard pattern in the image.
3. Refining the detected corners to sub-pixel locations.
4. Calibrating the camera using the sub-pixel corners.

The corner detection, sub-pixel refinement and calibration is performed using OpenCV. The calibration estimates the translation and rotation of the checkerboard in relation to the camera. The object coordinate system is placed at the corner of the checkerboard and defines the coordinates in object space as shown in figure 3.1. Then having defined the corners in object space, we can compare them to the detected corners in image space.

Stereo camera (extrinsic) calibration After the intrinsic parameters have been estimated we perform stereo camera calibration, where we estimate the cameras position and orientation in relation to each other. The stereo camera calibration is carried out in a similar manner as the single camera calibration, except that two cameras are used. Note that all the all the corners must be visible in both cameras. The intrinsic and distortion parameters from the single camera calibration are used and not changed during the calibration. As long as the internal parameters do not change and the stereo camera pair does not move relative to each other, it is sufficient to calibrate only once.

3.4 Quality assessment

Accurate calibration is essential for high-quality depth estimation and free-viewpoint rendering. In order to assess the quality of the calibration we need objective metrics.

A commonly used metric for evaluating the calibration quality is the reprojection error. The reprojection error is the pixel distance between a 3D point projected onto the image plane and the corresponding measured point in the image. However, his metric has been shown to be unreliable [25] [19] in that a low reprojection error does not necessarily mean accurate calibration. A better metric is the to use the rectification error [25] which directly impacts the quality of depth estimation and rendering. Consider a checkerboard visible in a camera pair. After calibration and image rectification, the epipolar constraint tells us that corresponding points should lie on the same scan-line. The rectification error is then the vertical distance between the rectified points. The merit of this metric is that it directly affects depth estimation because depth estimation usually assumes that the epipolar constraint is valid. However, for single camera calibration quality estimation, we are still limited to using the reprojection error since we only have one view of the checkerboard.

3.5 Experimental results

We want to ensure that our calibration is robust, and to find out how many views of the calibration pattern is neccessary in order to achieve robust calibration. To do this we do a series of different single camera calibrations with a varying number of views. We captured 100 views of a checkerboard pattern in different orientations in regard to the camera, with the rotation varying approximately between 0 and 45 degrees. We calibrate the cameras with a varying number of views, between 2 and 50. The views are selected randomly and for each number of views, we run 25 calibrations. We can then see how the estimated parameters change using a different number of views.

The results of these calibrations are shown as box-plots in figure 3.2, 3.5, 3.4, 3.5 for the estimated focal length, principal point, radial distortion coefficents and tangential distortion coefficients respectively. The horizontal axis represents the number of views used for the calibration, and the vertical axis show the estimated parameter as a box-plot of the 25 calibrations.

A clear tendency can be seen in all the box-plots, in that the parameters exhibit less variation as the number of views used for the calibration increase.

The focal length has more variation when we just use 2 views for calibration than

a hogher number, but the focal lengths remain relatively robust. Even for just 3 view calibration, the difference between the highest and lowest estimated focal lengths, is just around 100 pixels, which is approximately 3 percent of the total focal length. Estimation of the principal point however, varies to a larger degree. Even for 30 view calibration, we still have a difference of around 80 pixels between the largest and smallest principal point in both x and y directions. Even with 50 view calibration, the difference remains large, about 40 pixels. However, it is said in the literature that the principal point a) is not easily determinable and b) does not significantly impact geometric reconstruction as long as it remains within reasonable amounts (less than a quarter of the total image) for cameras with a moderate to small field of view [26] (i.e. they exhibit a small amount of radial and tangential distortion, which our lenses do).

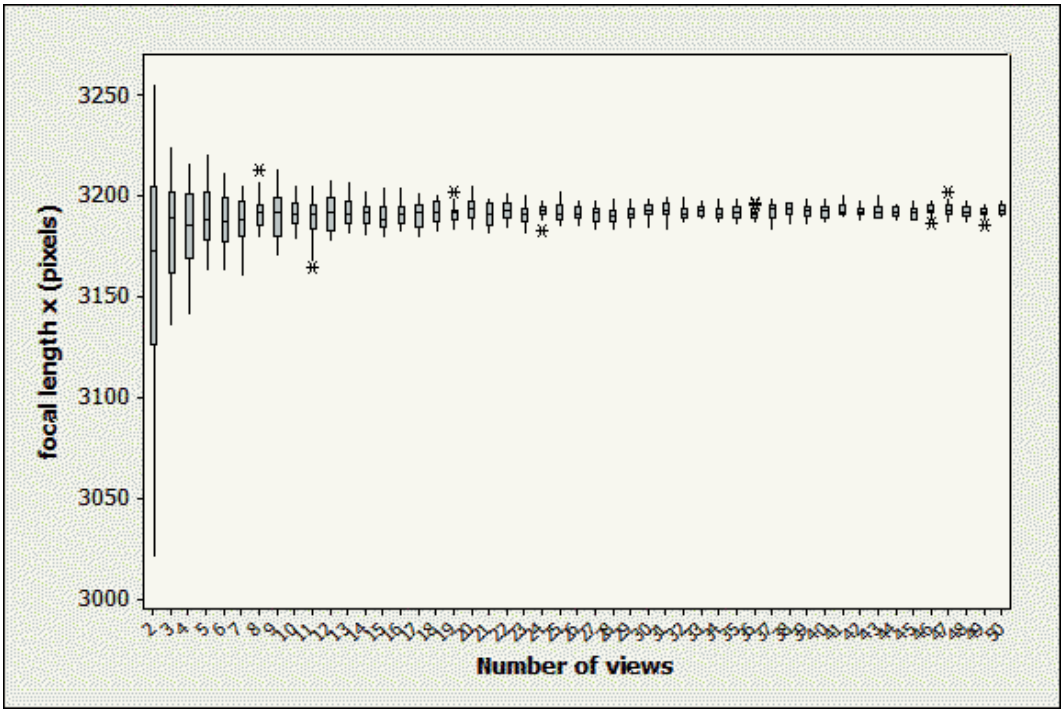
To verify the calibration we also compared the estimated parameters with that given by the manufacturer of the cameras and the lenses. The pixel size of the cameras are $3.75\mu m \times 3.75\mu m$, and the focal length of the lens is $12mm$. The estimated focal length is given in pixels, since we know the pixel width from the manufacturer we can find the estimated focal length in mm by

$$f = F \cdot pixels \cdot 3.75\mu m \tag{3.1}$$

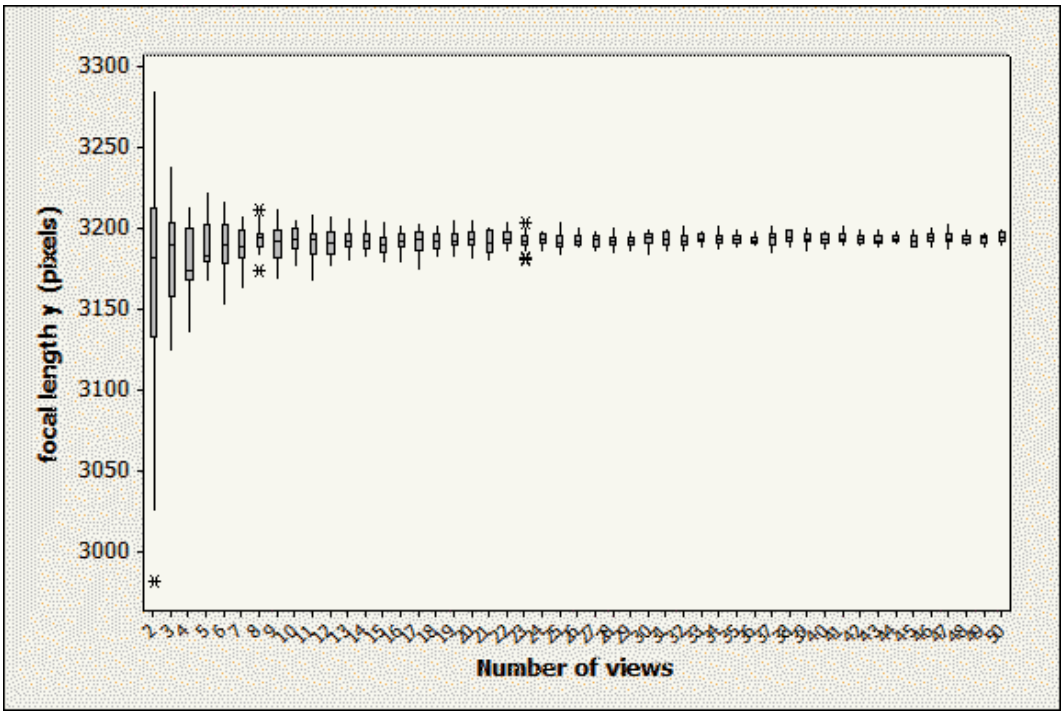
where F is the estimated focal length in pixels and f the focal length provided by the manufacturer in milimeters. Since we have done many calibrations with different estimated focal lengths, we instead find the focal length, given by the manufacturer in pixels. Solving equation 3.1 for F, we get

$$F = \frac{fmm}{3.75\mu m} = \frac{12mm}{3.75\mu m} = 3200pixels. \tag{3.2}$$

From figure 3.2 we see that our estimated focal lengths indeed lies close around 3200 pixels.

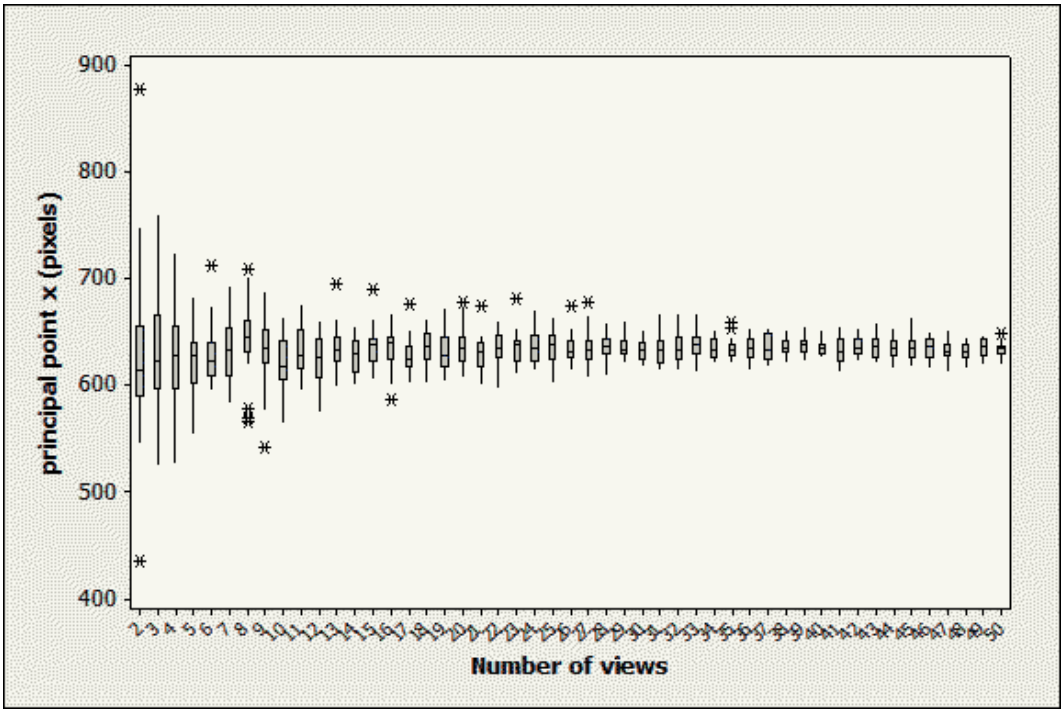


(a) Focal length x (pixels)

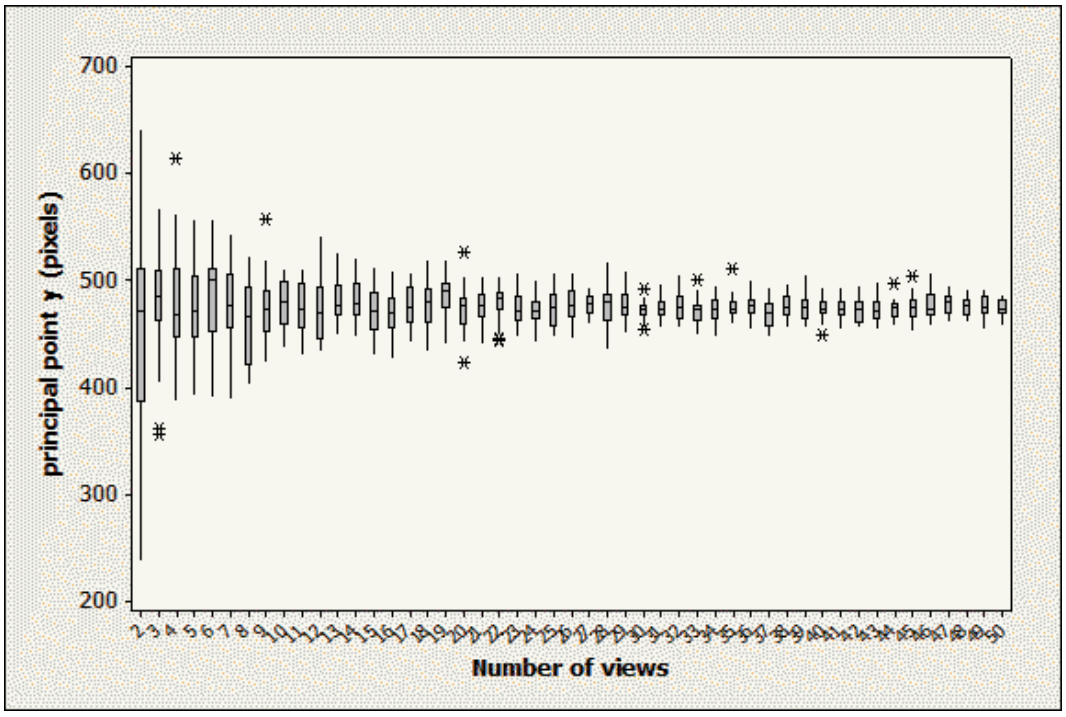


(b) Focal length y (pixels)

Figure 3.2: Box plots of estimated focal lengths.

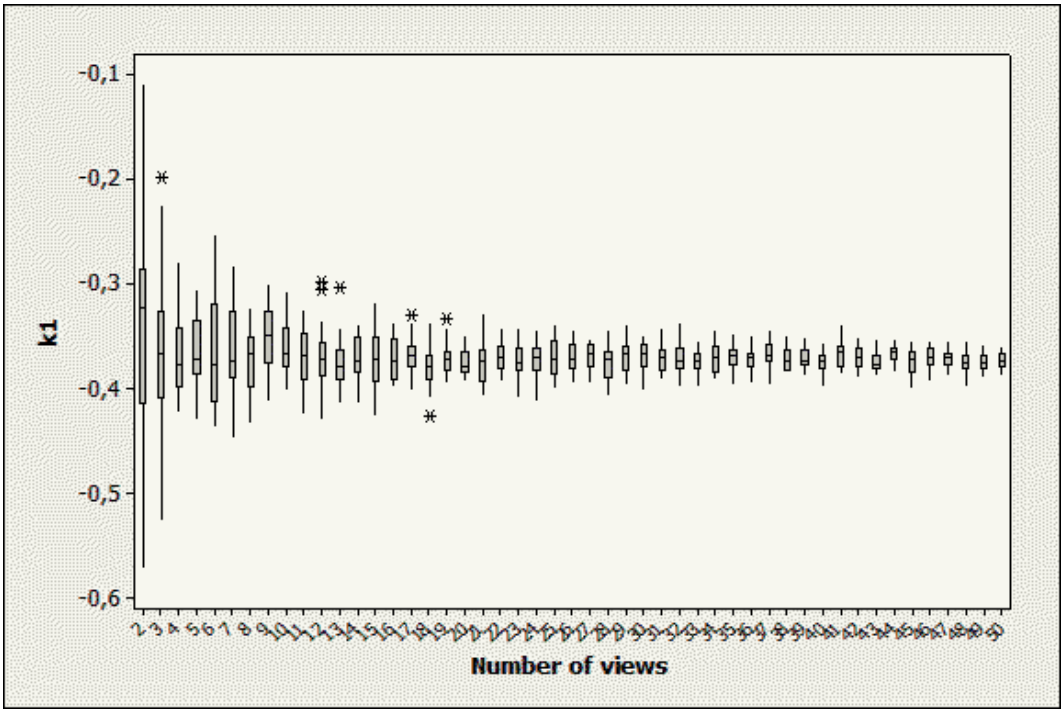


(a) Principal point x (pixels)

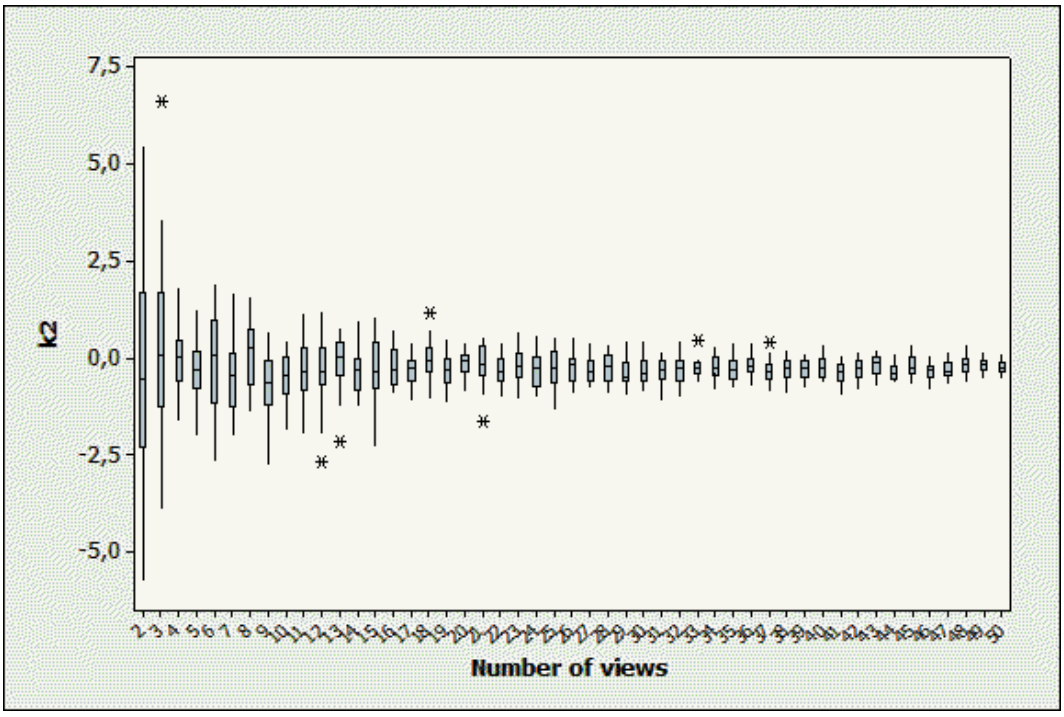


(b) Principal point y (pixels)

Figure 3.3: Box plots of estimated principal points.

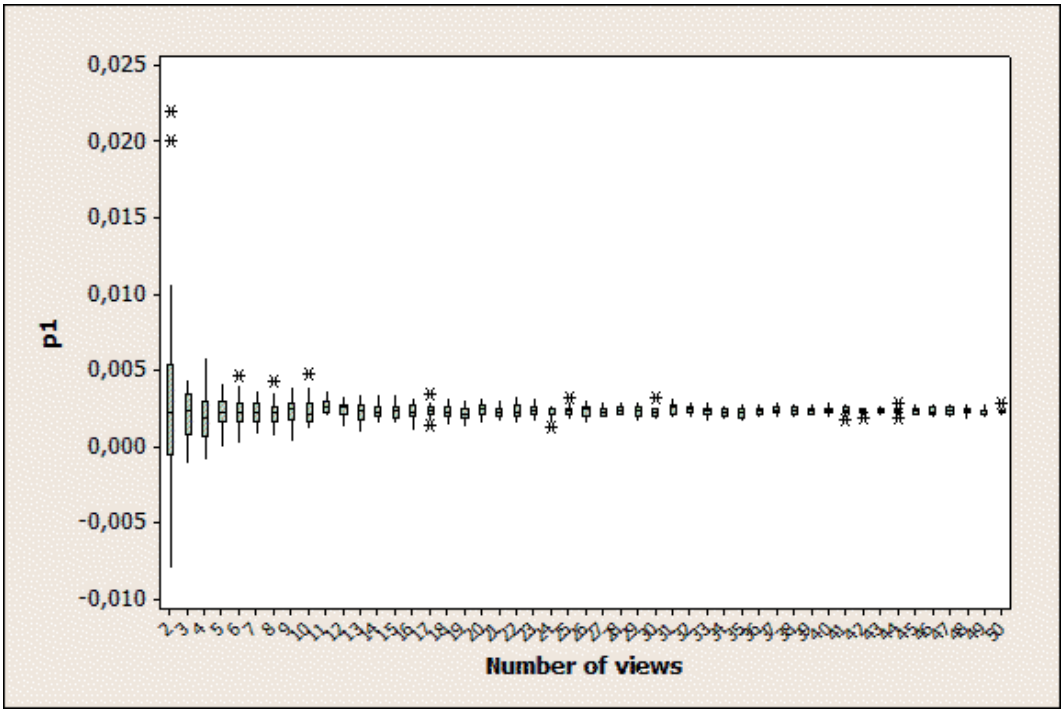


(a) Radial distortion coefficient k1

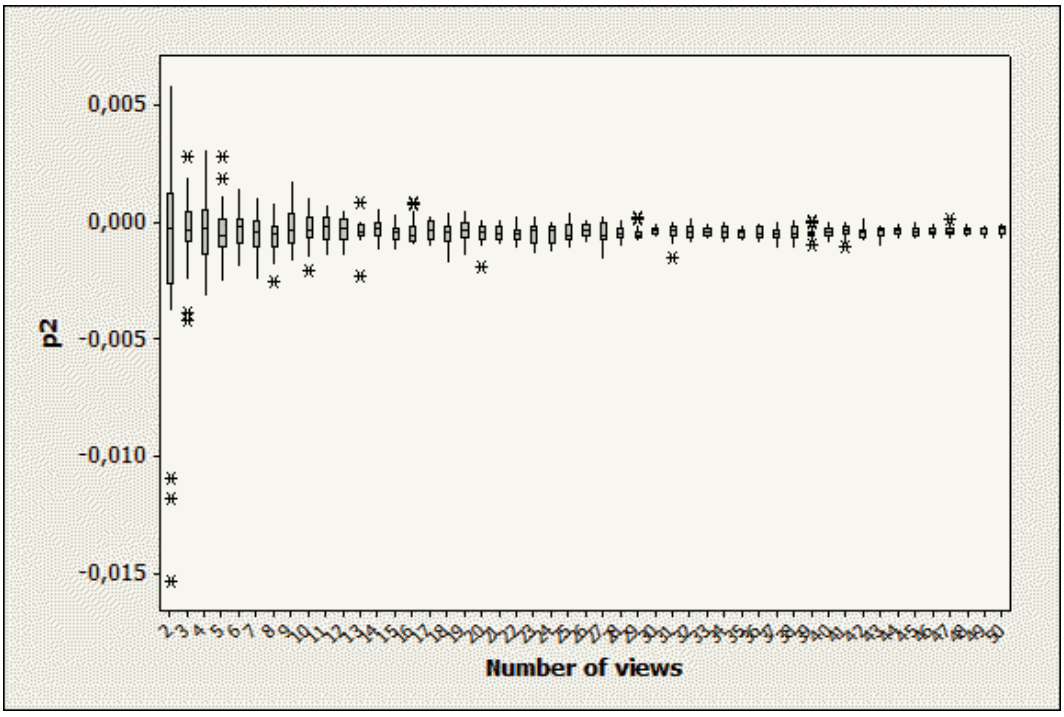


(b) Radial distortion coefficient k2

Figure 3.4: Box plots of estimated radial distortion coefficients.



(a) Tangential distortion coefficient p1

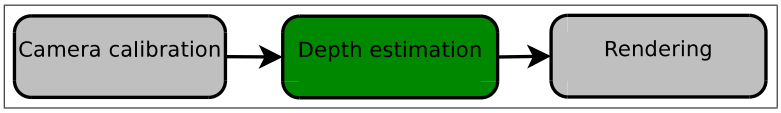


(b) Tangential distortion coefficient p2

Figure 3.5: Box plots of estimated tangential distortion coefficients.

Chapter 4

Depth estimation



4.1 Introduction

In depth estimation, we recover the depth of objects in a scene from images. In this thesis, we focus on binocular stereo, which estimates depth by finding corresponding pixels using two images from different vantage points. Consider two images of a scene taken from two different viewpoints where the epipolar lines are parallel and horizontal as illustrated in figure 4.1. For each feature in the left image, we search the right image for the same feature. If a feature pixel in the left view is at pixel coordinates (x_l, y_l) the corresponding feature pixel in the right view must, according to the epipolar constraint, lie on the same row and at the same column or to the left of that column [1]. The difference $d = x_l - x_r$ defines the disparity d which is a measure of coordinate difference of the same feature point in a stereo image pair.

[1] To demonstrate this effect you can hold up a finger in front of you and look at it with only the left eye. If you then look at your finger with only your right eye, the finger appears to shift left. Note that this effect decreases the further away the finger is from you. Objects closer to your eyes (or the cameras) have a large disparity and objects far from your eyes have smaller disparity.

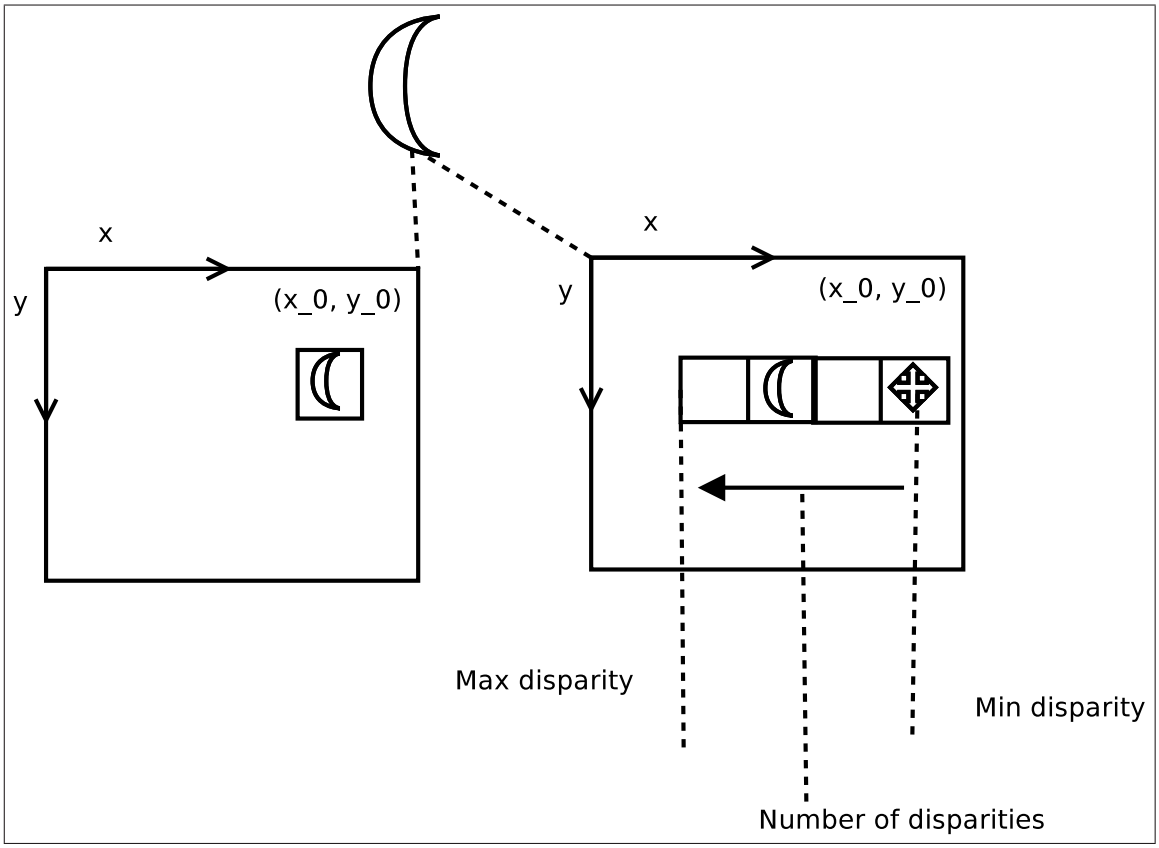


Figure 4.1: Disparity

The disparity can be calculated by finding corresponding pixels in the two viewpoints using color or intensity information. However, this is an ill posed problem. In texture-less areas with constant color, there is constant pixel correlation for any disparity value in that area, leading to unreliable disparity estimation. Corresponding points can also have different intensity values in the two images due to internal differences in the cameras. Even if the cameras would have identical internal properties, we would still have to assume (near to) lambertian surfaces, which reflects light uniformly in all directions, to ensure that a point has the same color from all vantage points. In objects such as mirrors or holograms, disparity can not be estimated since the color of the surface change according to the location of the camera. In addition, corresponding pixels might not be visible in both images, which is known as occlusions. Since pixel correspondences can not be found in occluded regions, no disparity value can

be calculated. It is apparent that depth estimation is hard, and that an intricate depth estimation procedure is required in order to estimate accurate depth values.

In section 4.2 we explain vital components of most disparity estimation algorithms and how the disparity values can be turned into depth values in section 4.3. In section 4.4 we present our depth estimation system. The depth estimation system consist of a pre-processing stage, a disparity estimation stage and a post-processing stage to turn the estimated disparity into depth. We then present methods to assess the quality of the estimated disparity/depth in section 4.5. Finally we present our obtained results in section 4.6.

4.2 Disparity estimation

Stereo correspondence algorithms can usually be broken down into four steps [27]:

1. Matching cost computation.
2. Cost aggregation.
3. Disparity computation/optimization.
4. Disparity refinement.

Matching cost computation The matching cost computation measures pixel correlation. Common measures are the Absolute Differences (AD) and Squared Differences (SD). However, the matching cost measure does not significantly impact the quality of the correspondence matching since the color channels are highly correlated, so using just the intensity does not significantly decrease the quality.

Cost aggregation Although single pixel matching are used in some stereo correspondence algorithms, the measures are often aggregated over a region around the pixel, e.g. using the sum of absolute differences (SAD) or the sum of squared differences (SSD), to obtain reliable matching. A large window gives fewer mismatches, especially at object surfaces with smoothly varying depth and texture-less regions, but gives unreliable results at object boundaries with discontinuous depth since the window contains both background and foreground pixels.

Disparity computation/optimization The disparity computation/optimization does the actual disparity value calculation. Local optimization methods rely mostly on the correlation measure and the cost aggregation. To compute the disparity, we simply choose for each pixel the disparity value that give the smallest cost. Such a strategy is known as "winner-take-all" (WTA) optimization. Global optimization methods calculates the disparity by minimizing a global energy function, selecting the disparity values that yield the lowest cost.

Disparity refinement If the disparity calculation is done for integer values, the disparity is limited to a discrete set of planes. Such a quantization is detrimental to virtual view synthesis, as the scene appears to be made out of thin shearing layers. Some algorithms therefore resort to refining the integer disparity into sub-pixel levels to increase the depth resolution. The disparity map can be further refined by median-filtering to remove high-frequency noise, cross-checking corresponding pixels from the left and right disparity maps to determine unreliable estimates and filling holes due to mismatches and occlusions.

4.3 Triangulation

The disparity is inversely proportional to depth, and a conversion between disparity and depth can easily be obtained by triangulation. Given the disparity $d = x_l - x_r$ between two corresponding points, where x_l and x_r are the pixel coordinates in the left and right view depicting a 3D point Q. The depth Z can then calculated by similar triangles:

$$\frac{T-d}{Z-f} = \frac{T}{Z} \implies Z = \frac{fT}{d} \tag{4.1}$$

with a baseline T and focal length f.

4.4 Implementation

In this section we present our depth estimation process, which is divided into three steps

1. Pre-processing
2. Disparity estimation
3. Post-processing

4.4.1 Pre-processing

We have to pre-process the images before the disparity estimation can be performed in OpenCV, that is we must

- Undistort the images to remove radial and tangential distortion.
- Rectify the images, so that the epipolar lines lie on horizontal scan lines.
- Modify the input images so that OpenCV also gives the right disparity map (the OpenCV disparity estimation functions just returns the left disparity map).
- Pad the input images so that OpenCV estimates disparity over the entire image.

Undistortion Since the captured images can suffer from radial distortion, they are undistorted prior to depth estimation. We have already estimated the radial and tangential distortions in the single camera calibration. The undistortion consist of a remapping of the pixels. We create an undistortion map, which contains a mapping from distorted pixel positions to undistorted pixel positions. This however, can lead to undefined pixels in the undistorted image. Therefore we use an inverse mapping that maps undistorted pixels positions to distorted pixel positions. We can then sample pixels from the distorted images and do resampling. This mapping is created once prior to the actual undistortion, and can be reused for each image frame that we undistort.

Rectification In addition to undistortion, the images are rectified prior to depth estimation, so that they have parallel and horizontal epipolar lines and identical intrinsic parameters. The rectification is a remapping (i.e. a transformation) of non-rectified images to rectified images. Since undistortion and rectification is just a sequence of two remappings, this remapping operation can be combined into a single remapping from distorted and non-rectified images to undistorted and rectified images.

Getting the right disparity map Unfortunately most OpenCV disparity estimation implementations only estimate the left disparity map from an image pair. For our implementation however, we need one depth map per image. Fortunately there are several ways to get also the right disparity map. 1) One approach is to switch the left and right images, and then setting a negative minimum disparity. Although, in the current implementation of OpenCV, this parameter can not be set for all the disparity estimation algorithms. 2) The disparity d is defined by $d = x_l - x_r$, where x_l and x_r are the left and right pixels positions respectively. Since we have the left disparity map we can calculate the right disparity map using $x_r = x_l - d$. Although this procedure can lead to hole artefacts in the right disparity map. 3) A more elegant solution is to get OpenCV to calculate also the right disparity without changing the actual implementation. This can be achieved by first flipping the two images around the vertical axis and then call the disparity estimation with the flipped right images as the left image and the flipped left image as the right image. This simple method is illustrated in figure 4.2.

Padding Unfortunately some of the OpenCV stereo functions does not calculate disparity over the entire image. The higher the range of possible disparities, the smaller the region of calculated disparities is. When the minimum disparity is zero, no disparity is calculated on the first $numberOfDisparities$ columns from the left side. To fix this we add a padding on the left side of the images, i.e. we add $numberOfDisparities$ pixels on the left side as seen in figure 4.3.

4.4.2 Disparity estimation

For estimating the disparity, we use readily available stereo correspondence methods in OpenCV, two CPU implementations and three GPU implementations using CUDA [28], namely:

- Block Matching (BM) on CPU [29]
- Semi-Global Block Matching (SGBM) on CPU [30]
- Block Matching on GPU
- Belief Propagation (BP) on GPU [31]

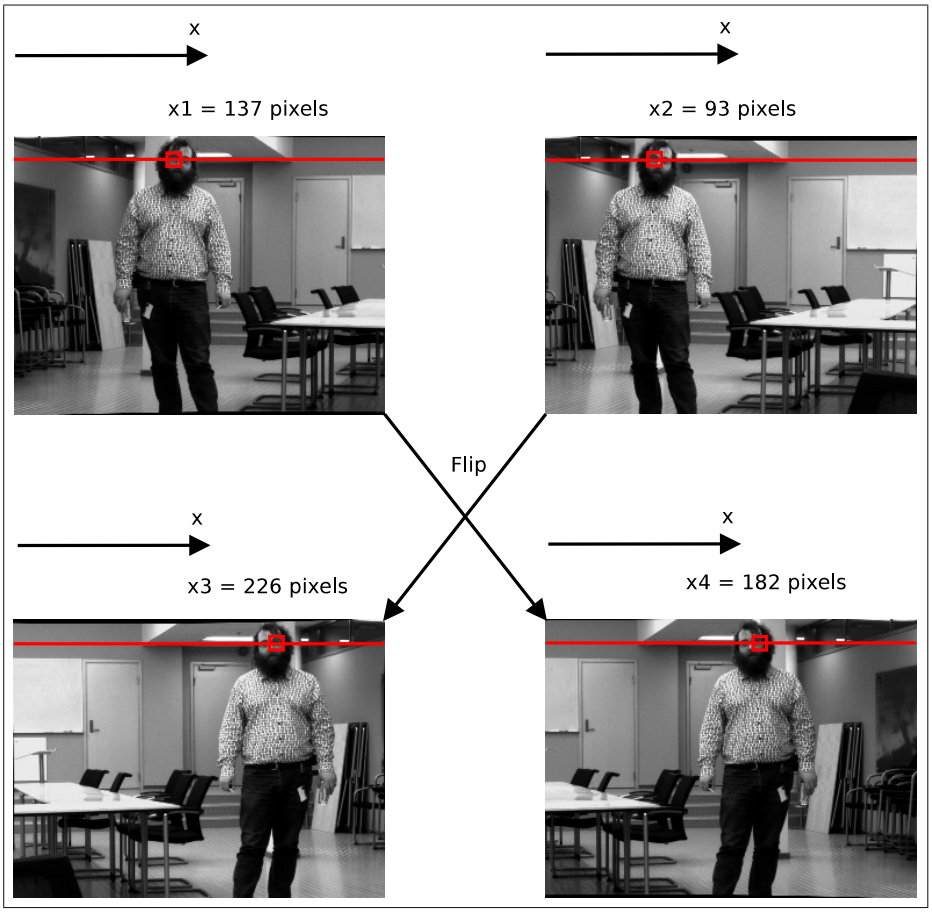
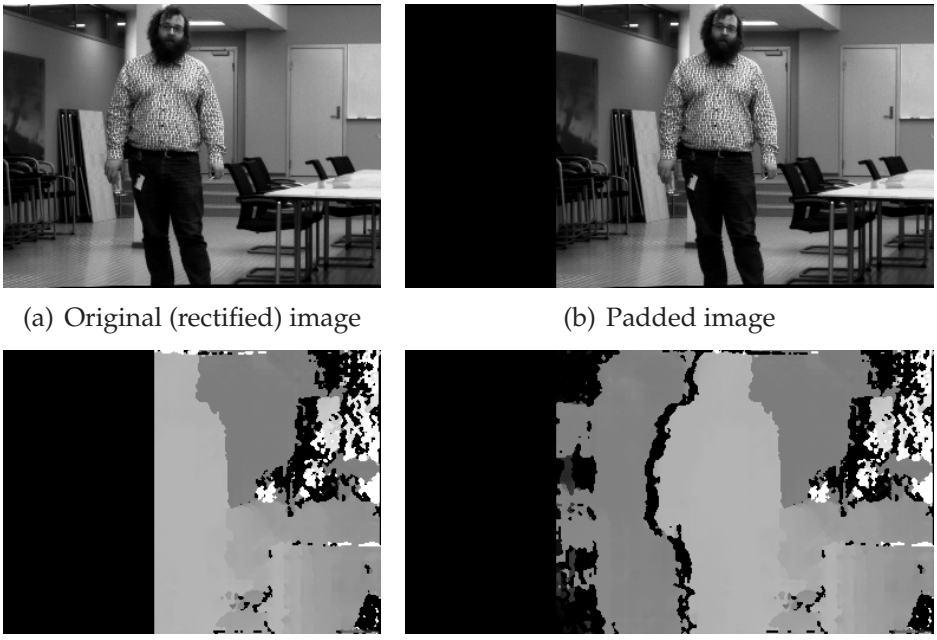


Figure 4.2: Simple way to get right disparity map in OpenCV. The input images are flipped around the vertical axis and the input images are switched (i.e. left flipped image is given as right input image and right flipped image is given as left input image).

- Constant Space Belief Propagation (CSBP) on GPU [32]

OpenCV also has a disparity estimation algorithm using graph cuts, but this method was not used in this thesis because of very long processing times which are not suited for real-time applications.

The BM implementations are local methods, using a sliding SAD window while SGBM approximates a global optimization. BP relies on global optimization, and CSBP is simply a less memory consuming version of BP.



(a) Original (rectified) image

(b) Padded image

(c) Estimated disparity without padding

(d) Estimated disparity with padding

Figure 4.3: In (a) we have an image without padding giving us the disparity map shown in (c). Observe that a disparity has not been calculated for a large portion of the image on the left side. In (b) we have padded the left side of the image which gives disparity values for a much larger portion of the image as seen in (d).

4.4.3 Post-processing

To get the final depth maps we perform several post-processing steps:

- Remove padding from the estimated disparity maps
- Flip the right estimated disparity map
- Scale the estimated disparities in case of sub-pixel estimation
- Triangulate the disparities to obtain the depth map
- De-rectify the depth maps to the original image planes.

Removing the padding For some of the disparity maps we added a padding on the left border. This padding should be removed after disparity estimation so that the

disparity map has the same size as the original images.

Flipping the right disparity For calculating the right disparity map we mirrored the images by flipping them around the vertical axis prior to disparity estimation. The estimated disparity map is therefore also mirrored, and is be flipped back around the vertical axis so that it corresponds to the right image.

Scaling Some of the disparity estimation algorithms find disparities at sub-pixel levels using integers. We therefore need to scale the disparities according to the number of sub-pixel disparities. To avoid losing precision we convert the disparities into floating point values.

Triangulation To convert the disparities into depth, we triangulate the disparities using equation 4.1 to obtain the depth values, which are the distances between the world coordinate system and the 3D-point along the Z-axis. Since the unit of the extrinsic parameters are given in the number of checkerboard corners, this is also the unit for the depth maps. If the size of the checkerboard corners were given at calibration however, metric reconstruction can be achieved.

De-rectification Since the images were rectified prior to disparity estimation, the disparity maps are projected back to their original image planes. OpenCV gives us the re-projection matrix for the left view, but the re-projection matrix for the right view must be set.

4.5 Quality assessment

Two commonly used approaches to evaluate the quality of depth estimation is either to compare the estimated depth map to ground truth data or to render a virtual viewpoint at the position of a reference camera using the estimated depth and then compare the synthetic image with the reference image.

[27] computes the root mean squared error (RMSE) between the estimated depth map and the ground truth map. They also calculate the percentage of bad matching

pixels, which is a percentage of the pixels which falls outside of a disparity threshold compared to the ground truth. In addition to calculating these values for the entire image, they divide the image into three regions, texture-less regions, occluded regions and depth discontinuity regions which are typical problem areas in depth estimation. [33] released a framework for evaluating depth estimation using laser scanned models.

4.6 Experimental results

To assess the quality of our disparity estimation system, we use data sets which are frequently used for assessing quality of disparity estimation algorithms. We therefore employ the datasets 'Tsukuba', 'Venus' [27], 'Cones' and 'Teddy' [34]. These data sets are provided with ground truth disparities and can be used by comparing them to our own estimated disparity maps. The ground truth images of 'Tsukuba', 'Cones' and 'Teddy' were acquired using structured lights while 'Venus' is composed of planar surfaces parallel to the image plane and ground truth disparity was hand drawn. We use the same evaluation as presented in [27], which consist of finding the number of bad pixels between the estimated disparity and the ground truth disparity. The absolute differences between the estimated and ground truth disparities are calculated, if the absolute value between the two pixels are over a given threshold, it is considered a bad pixel. The final score is given as a percentage of bad pixels in the entire disparity map. The metric is divided over three different kind of regions, discontinuous regions, occluded regions and 'all'. 'All' is the entire disparity map (except for Tsukuba which omit the pixels close to the boundaries of the disparity map). The discontinuous regions are regions with discontinuous depth, and occluded regions are regions which can not be seen by either one of the cameras. The reason for this division is that disparity can usually not be reliably estimated in occluded regions, and discontinuous regions tend to be more prone to errors than over smooth surfaces. The final evaluation criteria for each of the disparity estimation algorithms is then the average percentage of bad pixels for the four data sets.

The estimated left disparity maps are shown in figure 4.4 and the performance of the implementations are given in table 4.1 as a percentage of bad pixels. BM on the GPU clearly gives the worst results with a mean score of 41.6 % bad pixels, while BM

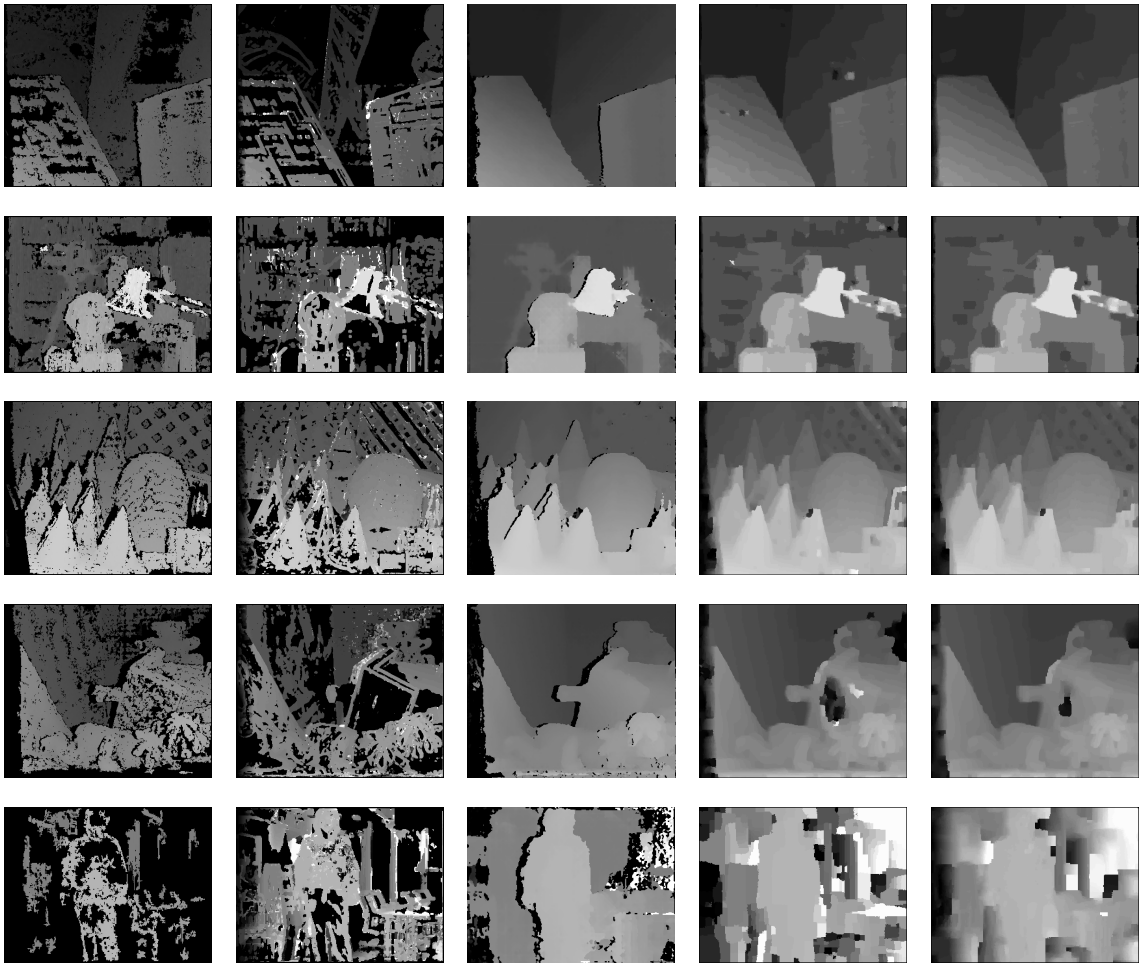


Figure 4.4: From left to right: BM on CPU, BM on GPU, SGBM, BP and CSBP. From top to bottom: 'Venus', 'Tsukuba', 'Cones', 'Teddy' and 'Man'.

on the CPU performs significantly better with 28.4 % bad pixels. It is also clear from visual inspection that both BM implementations contain a lot of holes and spurious disparity values in the GPU implementation (i.e. very high disparity values appearing as white in the disparity map). The remaining three implementations perform much better, with SGBM giving 12.3 % bad pixels, BP 10.8 % bad pixels and CSBP 10.4 % bad pixels. SGBM however, does not perform well on the discontinuous regions compared to BP and CSBP. That is also clear from the output disparity maps, where disparity values are often missing around the discontinuous regions.

The run time of the different disparity estimation algorithm was also tested to see

Table 4.1: Percentage of bad pixels for the different data sets, regions and algorithms.

	BM CPU	BM GPU	SGBM CPU	BP GPU	CSBP GPU
Tsukuba					
nonocc	20.0	42.0	4.00	2.58	2.54
all	21.8	43.2	5.61	4.23	4.27
disc	30.1	37.6	19.3	9.85	13.5
Venus					
nonocc	25.6	50.7	2.12	1.98	1.21
all	26.7	51.4	3.62	3.45	2.67
disc	35.6	44.5	22.9	17.8	16.2
Teddy					
nonocc	28.5	45.4	8.87	12.2	10.6
all	35.8	50.8	18.0	21.0	19.5
disc	41.5	41.9	22.8	15.4	19.0
Cones					
nonocc	16.0	26.4	6.25	7.70	5.82
all	25.5	34.4	16.3	17.3	15.7
disc	33.6	30.8	18.0	13.6	14.0
Average	28.4	41.6	12.3	10.8	10.4

how well they are suited to real-time applications. The tests were performed on an Intel Core i5-450M 2.4 GHz dual core processor with a NVIDIA GeForce GT 320M graphics card, which has 24 cores at 500 MHz. We used a 320x240 pixel image from our own captured data for the performance test. From the results in table 4.2 we see that the two BM implementations have the largest frame rates with 45 fps for the CPU implementation and 43 fps for the GPU implementation. SGBM also has a fairly high frame rate at 13 fps. BP and CSBP however, only has a frame rate of 1 fps and 0.15 fps respectively. Although, the graphics card is fairly low-end, so BP and CSBP would be expected to have significantly higher frame rates using a high-end graphics card. Still, on higher resolution images the BM and SGBM implementations might be the only viable options if a high frame rate is desired. BP and CSBP requires a significant amount of the graphics card's memory. On higher resolution images (e.g. 640x480 pixels), the

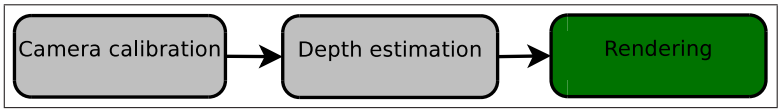
1 GB memory of our graphics card was not sufficient for the BP implementation. CSBP is more conservative on memory usage, but it still requires a significant amount.

Table 4.2: Frame rates of the different disparity estimation algorithms. Upload and download times between main memory and GPU memory is included.

BM on CPU	45 fps
BM on GPU	43 fps
SGBM on CPU	13 fps
BP on GPU	1 fps
CSBP on GPU	0.15 fps

Chapter 5

Rendering



5.1 Introduction

In section 5.2 we introduce how novel viewpoints can be generated using calibrated cameras and depth maps with a technique known as 3D warping. We continue in section 5.3 by explaining inherent artefacts in 3D warping and how those artefacts can be removed to achieve high-quality view synthesis. In section 5.4 we present our base rendering implementations and their variations. We review quality assessment of video and free-viewpoint rendering in section 5.5 and finish by presenting our experimental results in section 5.6. [1]

5.2 Rendering using 3D warping

To render synthetic viewpoints we use 3D image warping, which can render novel viewpoints using a reference texture image and its accompanying depth map. IBR

[1]To avoid confusing depth images/maps with color/intensity images, we also refer to images as textures.

techniques using such a depth map is called Depth Image Based Rendering (DIBR). In 3D warping we first back-project each pixel in the image to its 3D position using perspective projections, as explained in section 2.2.3, using the calibration data. The 3D point can then be projected onto the desired image plane, using the extrinsic and intrinsic parameters of the virtual camera (these values can be selected arbitrarily or computed from existing camera parameters). Let I_l and I_r be the textures from the left and the right camera. Let $p_l = (x_l, y_l, 1)^T$ and $p_r(x_r, y_r, 1)^T$ be the projections of a three-dimensional point $Q_w = (X_w, Y_w, Z_w, 1)^T$ through the cameras centers of projection C_l and C_r onto the left and right image planes. Given the point Q_w we can then find p_l and p_r by

$$\lambda_l p_l = \begin{bmatrix} K_l & |O_3 \end{bmatrix} \begin{bmatrix} R_l & t_l \\ O_3^T & 1 \end{bmatrix} Q_w \tag{5.1}$$

$$\lambda_r p_r = \begin{bmatrix} K_r & |O_3 \end{bmatrix} \begin{bmatrix} R_r & t_r \\ O_3^T & 1 \end{bmatrix} Q_w \tag{5.2}$$

where $\mathbf{R}_l$ and $\mathbf{R}_r$ is the rotation of the cameras, t_l and t_r is the position of the cameras and $\mathbf{K}_l$ and $\mathbf{K}_r$ are the intrinsic matrices. To synthesize a novel viewpoint we can then project the points of the source images onto the image plane of the novel viewpoint. Such a mapping from source pixels to destination pixels is known as forward warping.

5.3 High quality rendering

The simple approach presented in the previous section is not sufficient in order to render high-quality novel viewpoints. Specifically the following issues should be addressed:

- Visible surface determination
- Aliasing
- Small hole filling
- Disocclusion (large hole) filling
- Ghosting artefact removal

5.3.1 Visibility

When an image is warped, multiple source pixels can be mapped to the same pixel in the output image. This happens when a 3D point occludes another point in the synthesized view, which was visible in the original view. To resolve this, one must determine which pixel is visible, i.e. the point which is closest to the camera.

Z-buffering Z-buffering is a simple way to solve the visibility problem. A buffer is created for each pixel in the output image and initially set to infinity. When a pixel is warped to the new location, a depth comparison is performed between the pixel depth and the corresponding Z-buffer depth. If the newly warped pixel depth is less than that of the Z-buffer, the corresponding Z-buffer value is overwritten by the pixel depth value, ensuring that just the front-most pixel is rendered.

The painters algorithm Another approach to solve visibility is to apply the painters algorithm. The painters algorithm sorts points in a scene by their depth, and paints them back to front. Since background objects are painted first, foreground pixels overwrite background pixels, assuring correct ordering in the rendered view. However sorting the pixels by depth is computationally expensive. This can be solved using occlusion-compatible scanning order [8]. The procedure consists of projecting the camera center of the virtual viewpoint onto the source image plane and scanning along the epipolar lines, either from the image border towards the epipole or from the epipole towards the image border depending on the sign of the scale factor.

5.3.2 Aliasing

In general, the warping causes pixels to be mapped to sub-pixel locations in the novel viewpoint, but the pixels are discretized onto the image raster. Thus the new image is not correctly re-sampled, which can lead to aliasing. To avoid this the rendered view should be properly re-sampled, e.g. by using bilinear interpolation.

5.3.3 Small hole filling

Another artefact that can occur, due to the sampling rate of the image and differences in resolution of the input and output image, are undefined pixels in the rendered view. These sampling artefacts appears as black lines ("cracks") in the output image where no pixel value is set as illustrated in figure 5.1. These artefacts are visually disturbing and should be filled.

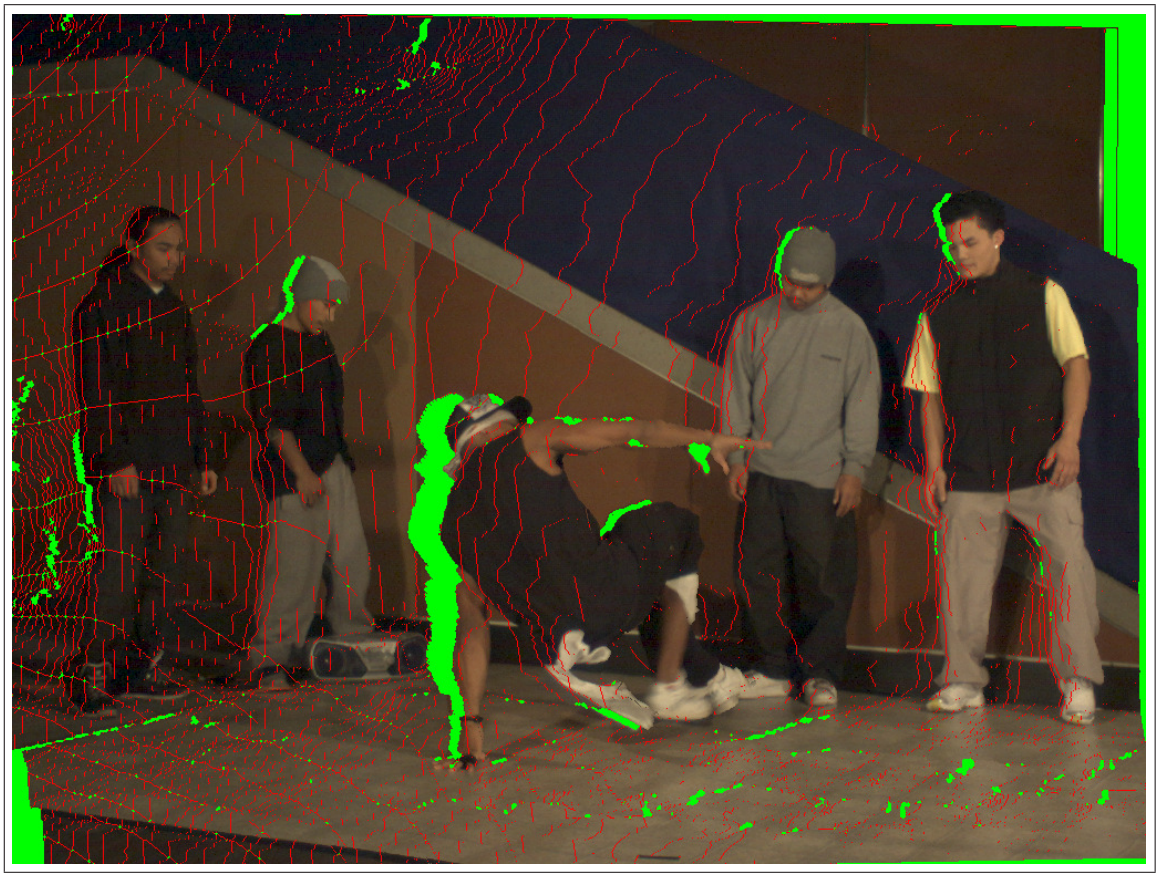


Figure 5.1: Warped image where cracks due to sampling rate is labeled with red and disoccluded regions labeled with green.

Oversampling One approach for removing the cracks is to oversample the image space. By doubling the resolution of the input image, most cracks in the rendered view are filled. However, the required amount of warping operations that need to be

performed is quadrupled. This is computationally expensive, especially considering that cracks are usually just a small percentage of the image.

Splatting The cracks can also be removed by using pixel splatting, which paints with a wider brush in the output image. Splatting maps a single pixel in the destination image to multiple pixels in the output image. Basically it increases the footprint of the warped pixel. The brush's size and shape depends on the spatial relationship of the source and destination images. It also depends on the distance of the camera to the three-dimensional image point. A disadvantage of splatting is that it can cause blurring of the image.

Inpainting A simple way to deal with the small holes is to inpaint them with color information from the surrounding pixels. This can be achieved by applying a simple interpolation from surrounding pixels. The cracks can be filled by a simple method such as nearest neighbor inpainting, but such an approach can create aliasing and other visible artefacts. A more complex method [35] creates a smooth transition. However, a proper labeling of cracks may be required to distinguish undefined pixels caused by the sampling rate and undefined pixels caused by disocclusions which is explained in section 5.3.4.

Inverse warping Another approach is using inverse warping [36] which use an inverse mapping that scans pixels in the output image and samples the input image. To back-project a pixel from the destination image onto the source image, we need the depth map of the novel view. To acquire this, the source depth map is first forward warped to the virtual viewpoint. Unfortunately the forward warping of the depth map suffers from the same artefacts as forward warping of the texture. The depth map however, is a low frequency image, since depth usually varies smoothly on object surfaces. Interpolating the cracks from surrounding pixels in the warped depth maps therefore leads to limited artefacts compared to warped textures. By using dilation [36] or a median filter [37] on the warped depth maps, cracks caused by warping are filled without filling in depth in disoccluded regions. Once the cracks are filled with depth values, the source texture can be sampled to synthesize the output image.

5.3.4 Disocclusions

Warping the image to a new location often creates large empty holes in the image as illustrated in figure 5.1, known as disocclusion. Disocclusions are areas which have not been seen by the source camera and are newly revealed in the novel viewpoint. The challenge is how to treat disocclusions correctly, since we do not have any data about the originally occluded regions in the source textures.

Inpainting The disoccluded regions can, like cracks caused by the sampling rate, also be inpainted using the color information from surrounding pixels. Although linear interpolation can fill in the disoccluded regions, it causes a visually disturbing "rubber-sheet" effect. Telea inpainting [35] cause less visually disturbing artefacts, but unfortunately both methods interpolate colors between foreground objects and background regions. The disoccluded regions are usually newly revealed background regions and an interpolation cause a smooth transition between background and foreground even if the two regions are discontinuous and should have a sharp border between them.

Depth based inpainting Instead of using an inpainting technique solely based on color information, the depth information can be used to find background regions [38] [] [39]. By using the forward warped depth map, the disoccluded regions can be inpainted by selecting color values from the background regions.

Multiple views Occlusions can be handled by forward warping two source images and composit them into a single rendered image [40]. This can fill in a large portion of the disoccluded regions, but usually not all undefined pixels are filled this way.

Static background If the background remains static, it can be captured prior to the dynamic scene, and later used to fill in disoccluded regions. Alternatively, the background can be constructed from multiple frames over time [41]. However, this requires that the background regions remains static and has constant illumination.

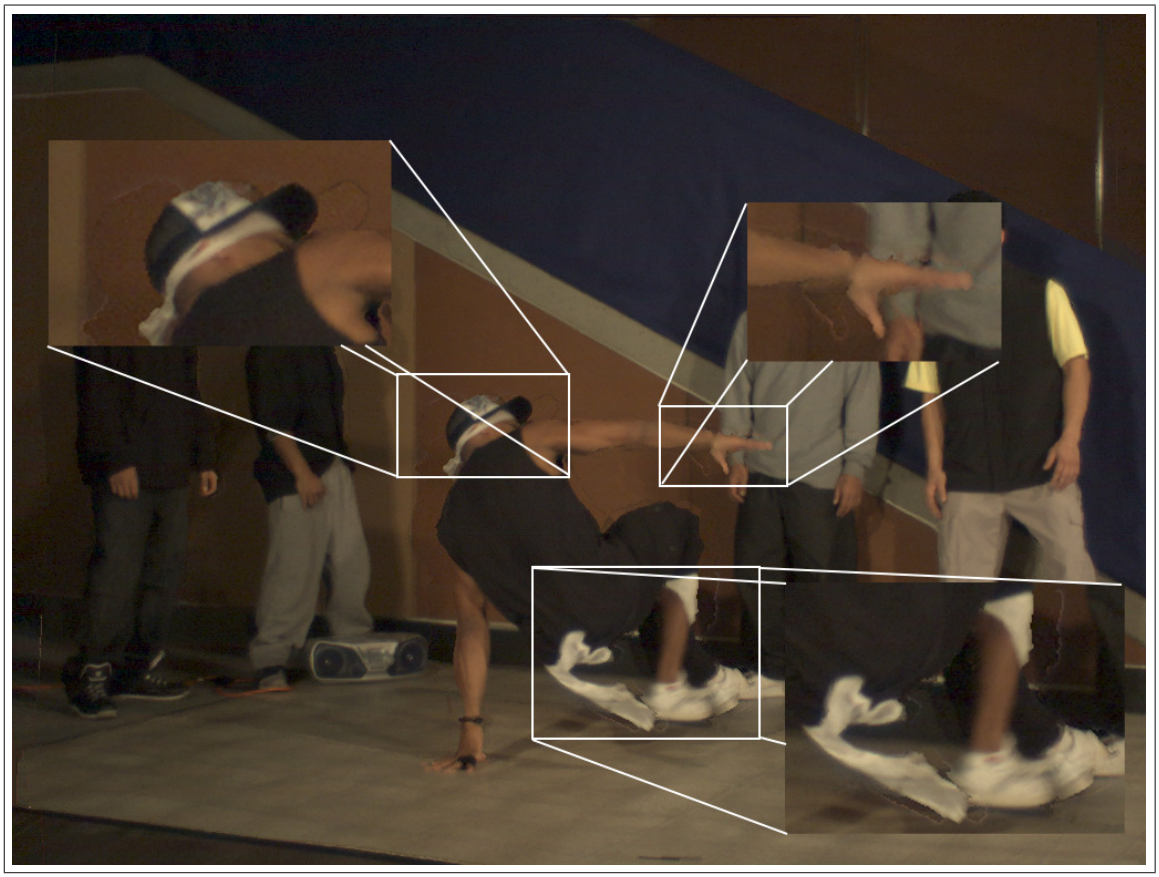


Figure 5.2: Ghosting artefacts around the foreground object in a synthesized view.

5.3.5 Ghosting artefacts

Most depth estimation algorithms are less accurate on the border between foreground and background objects. This happens because some points are not visible in both cameras and depth can therefore not be accurately estimated for those pixels. In addition, the internal construction of most digital cameras (i.e. the Bayer filter) cause some of the pixel color values to be interpolated from nearby pixels, leading to a blending of foreground and background pixels. Such pixels can not always clearly be labeled as background or foreground pixels. In depth maps the edges are usually sharp, while in textures the edges are more smooth, stretching over multiple pixels. This leads to foreground pixels being falsely identified as background pixels. When the image is warped, some foreground texture will remain at the background, leading to a "ghost"

around the object as shown if figure 5.2.

Eliminate ghosts by edge detection Since the ghosting artefacts usually occur around object boundaries, they should be treated separately. To do this edge detection is commonly used [37] [42] [17] [39] on the depth map to identify pixels in boundary regions. The boundary layers are typically excluded from the warping or otherwise independently processed.

5.4 Implementation

As we found no freely available implementations for free-viewpoint rendering, we implemented three different free-viewpoint rendering algorithms based on Mori et al. [37] (algorithm 1), Morvan [36] (algorithm 2) and Zinger et al. [39] (algorithm 3). All the three algorithms depends on inverse warping and the blending of multiple textures for high-quality free-viewpoint rendering. Some modifications were done, which are explained in the respective algorithm's presentation.

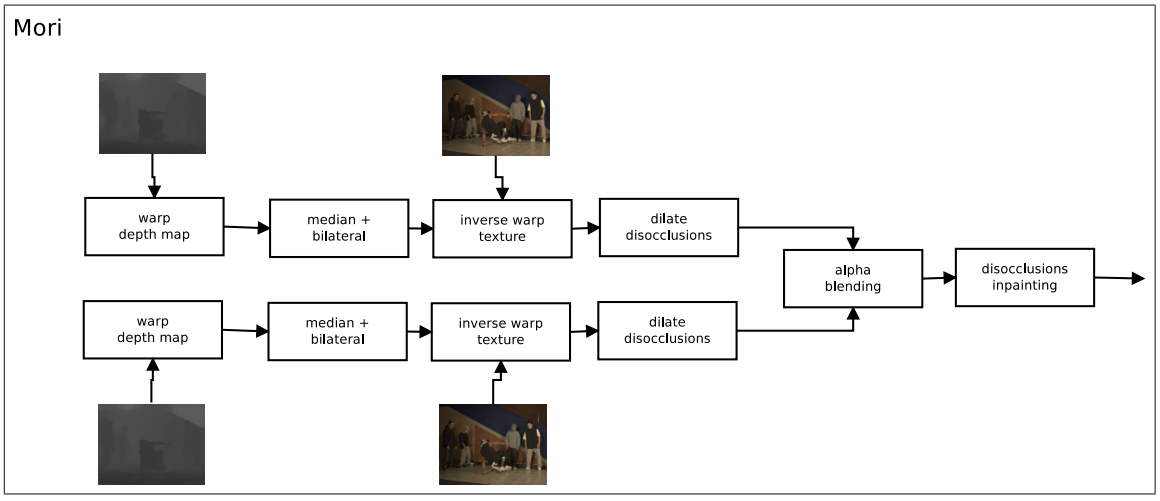
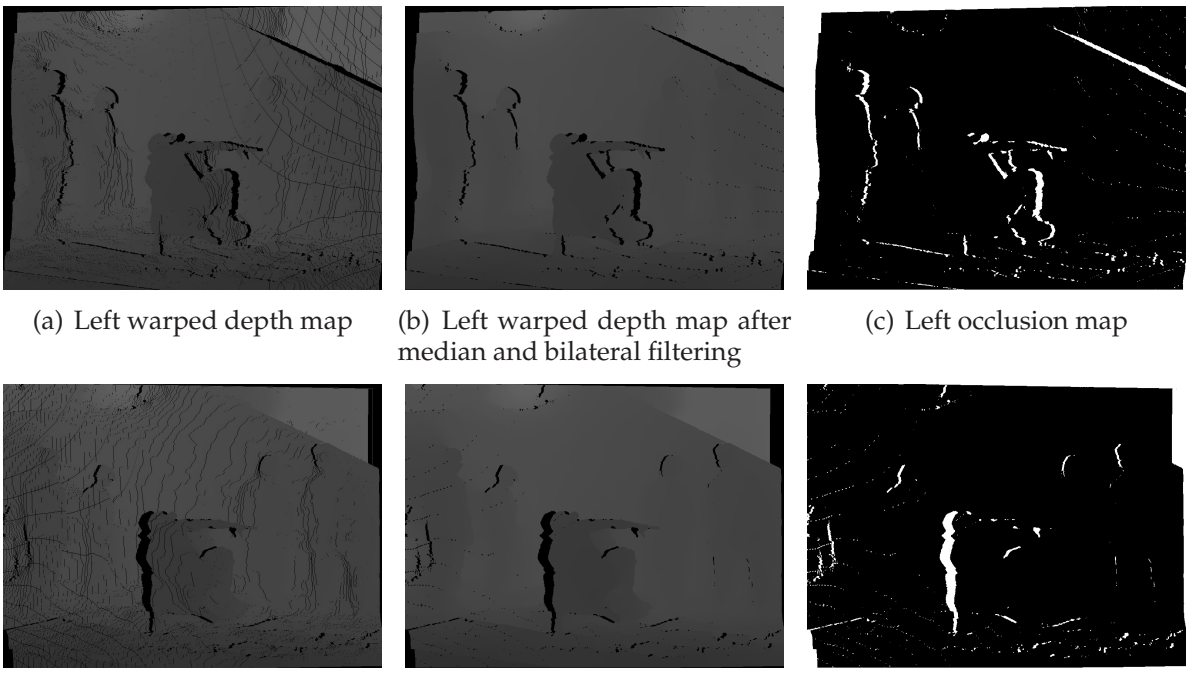


Figure 5.3: Pipeline for algorithm 1



(a) Left warped depth map

(b) Left warped depth map after median and bilateral filtering

(c) Left occlusion map

(d) Right warped depth map

(e) Right warped depth map after median and bilateral filtering

(f) Right occlusion map

Figure 5.4: In (a) and (d) we see the left and right depths maps after warping and in (b) and (e) the warped depth maps after median and bilateral filtering for algorithm 1. Observe that most of the small cracks in the depth maps are filled in. In (c) and (f) we see the remaining disoccluded regions.

5.4.1 Algorithm 1

Algorithm 1 has six steps, as shown in figure 5.3:

1. Forward warp depth maps
2. Median and bilateral filter the warped depth maps
3. Inverse warp textures
4. Dilate disocclusions (modified)
5. Blend textures
6. Inpaint disocclusions

Note that a texture and its corresponding depth map of a camera is processed independently of the other camera until the textures are blended.

Step 1 - Forward warp depth maps The source depth maps are initially warped to the novel viewpoint using forward warping into two separate buffers as shown in figure 5.4(a) and 5.4(d). Since multiple source pixels can map to the same pixel in the synthesized viewpoint, a Z-buffer is used to determine which pixel is visible to the virtual camera.

Step 2 - Median and bilateral filter depth maps A 3-by-3 median filter is then applied to the warped depth maps. The median filter fills a pixel with the median value of the nine surrounding pixels. The median filtered depth maps are then bilaterally filtered to smoothen the depth while preserving the depth values at edge pixels. We then have the filtered depth maps as shown if figure 5.4(b) and 5.4(e).

After the bilateral filter, some values in the disoccluded regions are filled with very low depth values. Therefore all the depth values under a given threshold are considered disoccluded and those pixels are erased from the depth maps. Let $depthLeft$ and $depthRight$ be the filtered depth maps where u and v are the integer pixel coordinates. When $depth[v][u] < threshold$, the pixel is considered occluded and set to zero. We then have

```
for (int v=0;v<height-1;v++) {
  for (int u=0;u<width-1;v++) {
    if (depthLeft[v][u] < threshold)
      depthLeft[v][u] = 0;
    if (depthRight[v][u] < threshold)
      depthRight[v][u] = 0;
  }
}
```

Listing 5.1: Find disoccluded pixels and erase low depth values caused from the bilateral filtering.

Note that the median and bilateral filter removed most of the small cracks, but the large holes (disoccluded regions) remain to be processed later .

Step 3 - Inverse warp texture Using the filtered depth maps, the source textures are then inverse warped by projecting each pixel of the depth map to its 3D-coordinate according to equation 2.12 and then projecting the point onto the source textures according to equation 5.1 or 5.2. The source textures are then sampled to create the warped textures shown in figure 5.5(a) and 5.5(b).

Step 4 - Dilate disocclusions As previously explained, ghosting artefacts appears at the background around disoccluded regions. To remove those artefacts the disoccluded regions are expanded. It is here assumed that the ghosting artefacts appear on the right side of the disocclusions for the left warped texture and on the left side for the right warped texture. For each disoccluded pixel $depth[v][u] = 0$ we search for the first non-zero depth pixel $depth(u, v \pm w)$ with offset w in the direction the ghosting artefacts are assumed to appear and mark it as disoccluded. Instead of just marking a 1 pixel wide border as disoccluded as in [37], we mark a 2 pixel wide border to remove additional ghosting artefacts.

```
for (int v=0;v<height-1;v++) {
  for (int u=0;u<width-1;v++) {

    if (depthLeft[v][u] == 0) { //disoccluded
    disoccludedLeft[v][u] = true;
    for (int w=0;u+w<=width-1;w++) {
      if (depthLeft[v][u+w] > 0) {
        disoccludedLeft[v][u+1] = true;
        disoccludedLeft[v][u+2] = true;
        break;
      }
    }
  }

  if (depthRight[v][u] == 0) { //disoccluded
    disoccludedRight[v][u] = true;
    for (int w=0;u-w>=0;w++) {
      if (depthRight[v][u-w] > 0) {
        disoccludedRight[v][u-1] = true;
        disoccludedRight[v][u-2] = true;
        break;
```

```
            }
        }
    }
  }
}
```

Listing 5.2: Dilation of the disoccluded areas.

(a) Left inverse warped texture (b) Right inverse warped texture (c) Blended textures

Figure 5.5: In (a) and (b) we see the warped left and right textures for algorithm 1. The textures are then blended giving us the image shown in (c).

Step 5 - Blend textures The two warped textures are then composited into a single viewpoint. If both pixels are disoccluded it is added to the inpaint mask to be filled in later. When a pixel from the left texture is marked as disoccluded we use the pixel from the right texture and vice versa. If none of the pixels are disoccluded, they are blended. The blended pixel value is determined by a weighted average of the two color values, so that the source camera which is closest to the virtual viewpoint is weighted more. The weighting coefficient α is calculated by

$$\alpha = \frac{|t - t_l|}{|t - t_l| + |t - t_r|} \tag{5.3}$$

where t is the translation vector of the virtual camera and t_l and t_r are the translation vectors of the left and right cameras respectively. The blending is then performed as follows:

```
for (int v=0;v<height-1;v++) {
  for (int u=0;u<width-1;v++) {
    if (disoccludedLeft[v][u] && disoccludedRight[v][u]) {
```

```
      imageBlended[v][u] = (1-a)*imageLeft[v][u]+a*imageRight[v][u];
    } else if (! (disoccludedLeft[v][u] && disoccludedRight[v][u]) {
      inpaintMask[v][u] = true;
    } else if (disoccludedLeft[v][u]) {
      imageBlended[v][u] = imageRight[v][u];
    } else {
      imageBlended[v][u] = imageLeft[v][u];
    }
  }
}
```

Listing 5.3: Blending function

(a) Inpaint mask (b) Inpainted texture

Figure 5.6: The inpaint mask shown in (a) defines the final disoccluded regions after blending, where we do not have any texture values. These regions (marked in white) are inpainted giving us the texture in (b).

Step 6 - Inpaint disocclusions Finally the remaining holes in the image defined by the *inpaintMask* (figure 5.6(a)) are inpainted using the method of Telea [35]. This gives us the blended texture as shown in figure 5.6(b).

5.4.2 Algorithm 2

Algorithm 2 generates the novel viewpoint in five steps:

1. Forward warp depth maps
2. Dilate and erode the warped depth maps

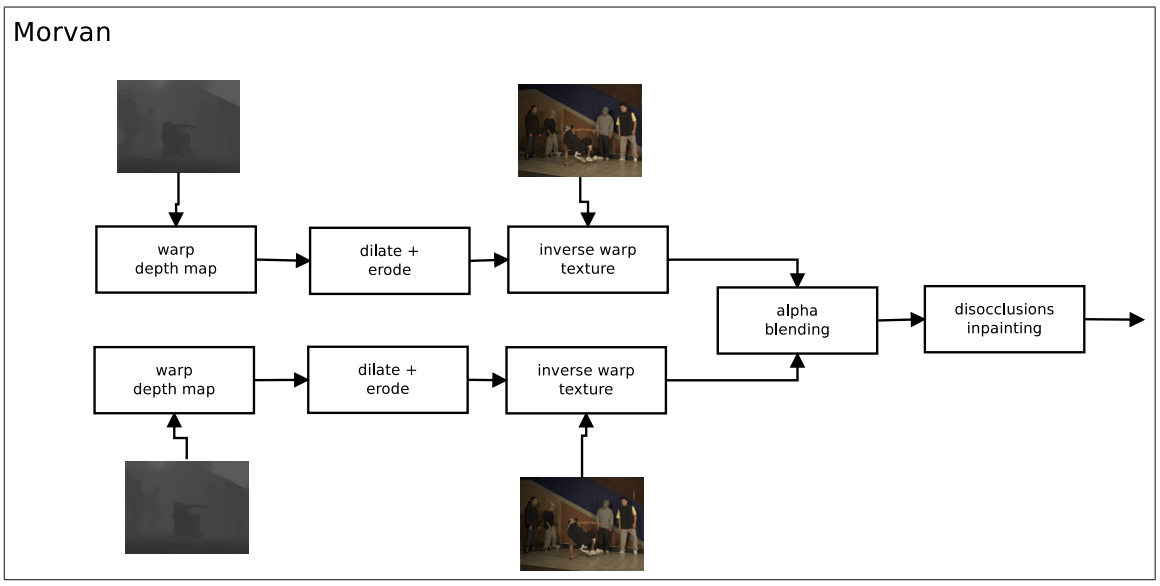
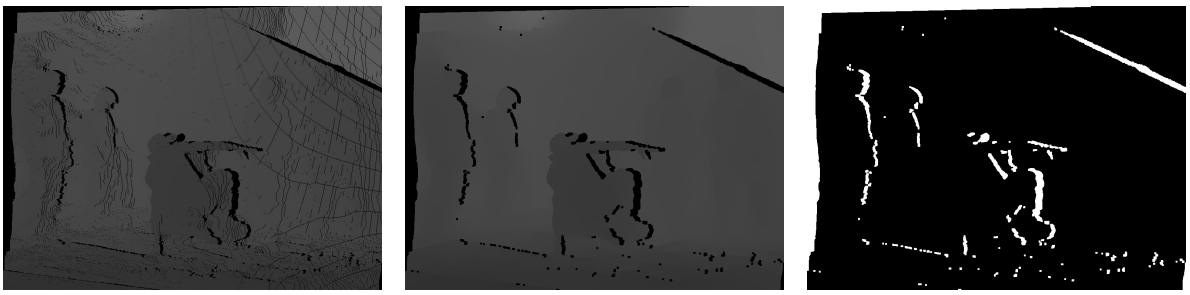


Figure 5.7: Pipeline for algorithm 2

3. Inverse warp textures
4. Blend textures
5. Inpaint disocclusions (modified)

Step 1 - Forward warp depth maps First the source depth maps are forward warped to the virtual viewpoint similar to step 1 in Algorithm 1.

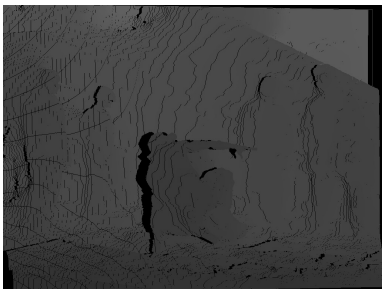
Step 2 - Dilate and erode the warped depth maps Instead of filling the small holes by using a median filter, they are filled using two dilation and one erosion operation. The dilation operator fills a pixel with the maximum value of the surrounding pixels while the erosion operator fills a pixel to the minimum value of the surrounding pixels. For both the dilation and erosion we use a 3-by-3 window. We then have the processed depth maps as shown in figure 5.8(b) and 5.8(e). These operations not only fills cracks in the warped depth maps, but actually expands the border of the foreground objects which are not bordering disoccluded areas. The disocclusions are also expanded, and erase some of the ghosting artefacts, since they tend to be at the background area bordering the disoccluded regions.



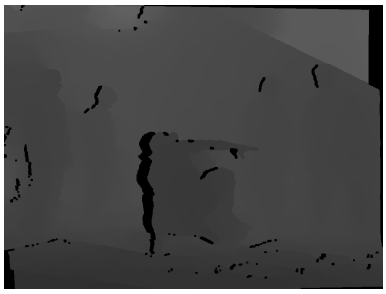
(a) Left warped depth map

(b) Left warped depth map after two dilations and one erosion

(c) Left inpaint mask



(d) Right warped depth map



(e) Right warped depth map after two dilations and one erosion

(f) Right inpaint mask

Figure 5.8: In (a) and (d), we see the warped left and right depth maps for algorithm 2, and the post-processed depth maps in (b) and (d). (c) and (f) shows the disoccluded regions which are to be inpainted.

Step 3 - Inverse warp texture: The source textures are then inverse warped similar to algorithm 1 step 3, using the dilated and eroded depth maps. Since the inverse warping generally gives us sub-pixel coordinates in the source views, it allows us to do a simple bilinear resampling, which takes a weighted average of the four surrounding pixels in the source image. The pixels which does not have a defined depth value are labeled as occluded in an inpaint mask, which define the regions which should be inpainted.

Step 4 - Inpaint disocclusions If neither of the pixels are defined, it is extrapolated from the background color utilizing the epipolar geometry. It was not clear how Morvan's technique of padding disocclusions [36] could be performed using occlusion-compatible scanning order when inverse warping is used. Instead, we implemented a different technique which also relies on the epipolar geometry. First the source camera center is projected onto the virtual viewpoint. The 3D point of the camera center is the

(a) Left texture inpainted (b) Right texture inpainted

(c) Blended texture

Figure 5.9: In (a) and (b), we see the inpainted left and right textures for algorithm 2, and the blended texture in (c).

translation vector of that camera, and this is projected onto the virtual viewpoint using the projection matrix of the virtual camera. The line formed by a disoccluded pixel and the projected camera center (i.e. the epipole) forms the epipolar line. This line is extended so that it spans over the entire image. Starting at the disoccluded pixel, we search in both directions (i.e. towards and away from the epipole) along the epipolar line to find the first pixels on both sides with a non-zero depth value. The line between those pixels then covers only disoccluded pixels. A depth comparison is performed on the two pixels on both sides of the disoccluded region, choosing the one with the highest depth value, so that background pixels are preferred over foreground pixels. The color value of the pixel with the highest depth is then used to fill in the line covering the disoccluded region. This process is repeated for each disoccluded pixel until they are all filled with color values.

Step 5 - Blend textures The warped textures are then composited into the virtual viewpoint. If the corresponding pixels colors are similar (i.e. the difference is under

a given threshold) they are assumed to be from the same 3D point and are blended by calculating the mean color values of the two pixels. If the colors are inconsistent a depth comparison is performed and the front-most pixel is used. If a pixel from one view is undefined on the other hand, the pixel from the opposite view is used. Finally, if neither pixel is defined the inpainted pixels from the two views are blended.

5.4.3 Algorithm 3

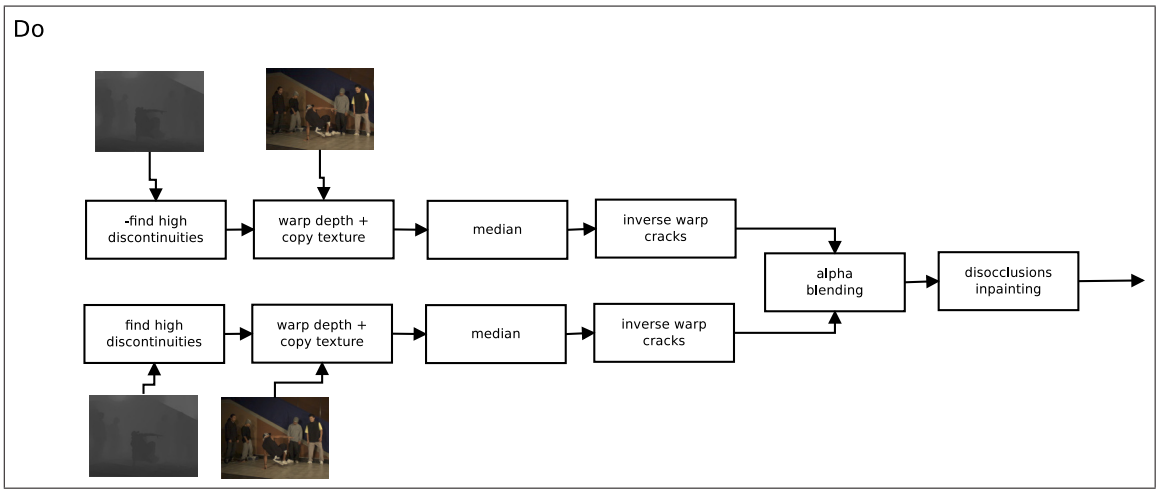


Figure 5.10: Pipeline for algorithm 3

Algorithm 3 is performed in 6 steps as illustrated in figure 5.10, namely:

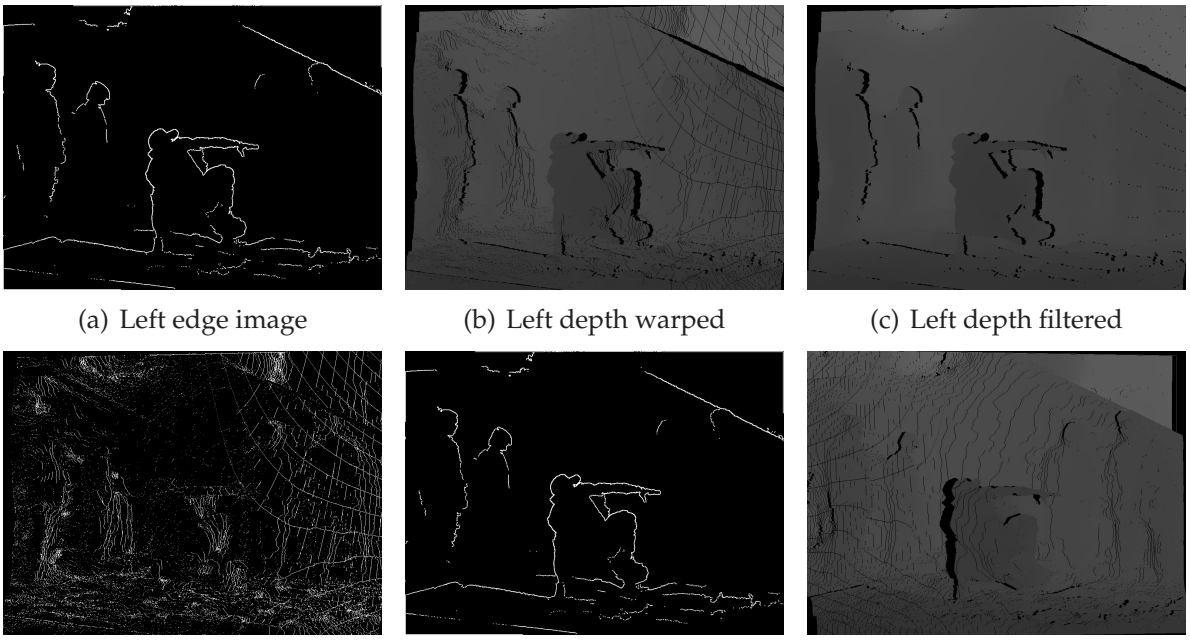
1. Detect edges (modified)
2. Forward warp depth maps and textures
3. Median filter depth maps
4. Inverse warp cracks
5. Blend textures
6. Inpaint disocclusions

Step 1 - Detect edges To avoid ghosting artefacts the pixels near high depth discontinuities (i.e. edges) are not warped. Edge pixels in the two views are identified by summing the nine depth values around the pixel. If the difference between nine times the pixel depth value of the center pixel and the summed depth value is greater than a given threshold (i.e $9 \cdot depth - sum > threshold$), the pixel is considered a ghosting artefact and omitted from the warping in the next step.

```
bool isEdge(float depth[][], int u, int v) {
  double threshold = 10.0;
  double sum = 0.0;

  for (int i=-1; i<=1; i++) {
    for (int j=-1; j<=1; j++) {
      sum += depth[v+j][u+i];
    }
  }

  if (9*depth[v][u]-sum > threshold)
    return true;

  return false;
}
```

This approach differs slightly from Zinger et al. [39], which label edges where $sum - 9 \cdot depth > threshold$. They expand the labeled edges by one pixel, but do not explain how such an expansion is performed other than that "... an additional labeling step is required." [43]. The former (i.e. our) labeling step labels background pixels near edges while the latter (i.e. Zinger et al.) labels foreground pixels near edges. To expand the edge regions we also labeled the eight surrounding pixels as edges, so that it forms a 3 pixel wide border. If we initially had detected edges at the foreground, the expansion of the edges would have removed foreground pixels instead of the ghosting artefacts at the background. Figure 5.11(a) and 5.11(e) shows examples of such an edge detection.

Step 2 - Forward warp depth maps and textures The source depth maps and textures which are not marked as edge pixels are forward warped.



(a) Left edge image (b) Left depth warped (c) Left depth filtered

(d) Left changed pixels (e) Right edge image (f) Right depth warped

(g) Right depth filtered (h) Right changed pixels

Figure 5.11: In (a) and (e) we see the detected edges in the left and right depth maps. (b) and (f) shows the depth maps after warping with the edges omitted, and the filtered depth maps can be seen in (c) and (g). Finally in (d) and (h), the pixels that changed after median filtering is marked in white.

(a) Left texture forward warped

(b) Left texture after inverse warping cracks

(c) Right texture forward warped

(d) Right texture after inverse warping cracks

Figure 5.12: In (a) and (c) we see the forward warped left and right textures. (b) and (d) show the same textures after the cracks have been filled by inverse warping.

Step 3 - Median filter depth maps To fill in the cracks in the depth maps caused by the forward warping, they are processed with a median filter similar to algorithm 1, except that a 3-by-3 filter is used and no bilateral filtering is done. We then find the absolute difference between the warped depth maps and the median filtered depth maps. The binary difference image of this operation is shown in figure 5.11(d) and 5.11(h).

Step 4 - Inverse warp cracks The textures are then inverse warped, but only those pixels that changed during the depth map median filtering. The warping then fills in the cracks caused by the initial forward warping of the textures. We then have the textures shown in figure 5.12(b) and 5.12(d).

Step 5 - Blend textures The warped textures are blended similar to algorithm 1, except that not all pixels are blended this way. First a soft depth comparison is performed using the two filtered depth maps. Unlike a hard depth comparison, the front-most pixel is not always chosen to be rendered. Instead we first check if the depth difference lies under a small threshold (i.e. the depth values are near to each other). If they are, they are blended as before, if not the front-most pixel is rendered instead.

Step 6 - Inpaint disocclusions Finally the remaining undefined pixels after blending are filled using a depth based inpainting technique. For each occluded pixel we search in eight directions for the first defined depth pixel. We then use the N depth pixels which has the smallest depth (i.e. pixels at the background). The pixels are then blended according to equation 5.4:

$$\frac{\sum_{i=1}^{N} m_i^{-2} t_i}{\sum_{i=1}^{N} m_i^{-2}} \tag{5.4}$$

where m_i is the distance from the occluded pixel to the edge of the disocclusion, and t_i is the color value of the texture.

(a) Blended texture

(b) Blended depth

(c) Inpaint mask

(d) Inpainted texture

Figure 5.13: (a) shows the blended texture and (b) the blended depth. The pixels with zero depth in the blended depth defines the inpaint mask shown in (c). The inpaint mask defines the pixels which should be inpainted, and a search in the blended depth is performed to find the nearest pixels with the lowest depth values. After inpainting we get the image shown in (d).

5.4.4 Variations of the base implementations

In addition to the base implementations (algorithm 1, 2 and 3), we implemented a large number of variations to improve the quality of the rendered viewpoints. [2]

No bilateral filter We removed the bilateral filtering of the warped depth maps in algorithm 1 step 2.

5x5 median filter We replaced the 3x3 median filter with a 5x5 median filter in algorithm 3 step 3.

Dilate disocclusions We added dilation of the disoccluded regions for all three base implementation. This expansion of the disoccluded area is done in all directions, not just in the left and right direction as performed in algorithm 1 step 4. Expanding the disoccluded regions in this way can also remove good pixels in addition to the "ghosting" artefacts. To deal with this we present a novel cross-checking constraint for erasing pixels around disoccluded areas. The cross-checking constraint only allows erasing pixels which are visible in at least one of the two warped images.

We replaced the disocclusion dilation in algorithm 1 step 4 with:

- 2 pixel disocclusion dilation.
- 2 pixel disocclusion dilation with cross-checking.

In algorithm 2 we dilated the disoccluded regions between step 3 and 4, i.e. :

- 1 pixel disocclusion dilation.
- 1 pixel disocclusion dilation with cross-checking.

Algorithm 3 omits warping a 3 pixel wide border around edges. We replace this with detecting a 1 pixel wide border and with different variations of dilating the disocclusions. We also add 5x5 median filtering of the warped depth maps and disocclusion cross-checking.

[2] These variations were implemented as a direct result of the artefacts in the base implementations presented in section 5.6. For completeness, the implemented variations are presented in this section.

- 1 pixel edge detection and 1 pixel disocclusion dilation.
- 1 pixel edge detection and 1 pixel disocclusion dilation with 5x5 median filtering.
- 1 pixel edge detection and 1 pixel disocclusion dilation with cross-checking.
- 1 pixel edge detection and 1 pixel disocclusion dilation with cross-checking and 5x5 median filtering.

We then remove the edge detection completely and instead perform the above variations, except that 2 dilation operations are performed to expand the disocclusions by 2 pixels, i.e. :

- No edge detection and 2 pixel disocclusion dilation.
- No edge detection, 2 pixel disocclusion dilation and disocclusion cross-checking.
- No edge detection, 2 pixel disocclusion dilation and 5x5 median filtering.
- No edge detection, 2 pixel disocclusion dilation, disocclusion cross-checking and 5x5 median filtering.

Bilinear interpolation Algorithm 2 performs bilinear interpolation on the inverse warped textures. We also added this to algorithm 1 in step 3 and to algorithm 3 in step 2 and 4. Note that in the base implementation of algorithm 3, textures are forward warped in step 2. To perform bilinear interpolation in this step we first forward warp depth, and then inverse warp the texture with bilinear interpolation.

Zinger inpainting The inpainting in algorithm 1 step 6 and algorithm 2 step 4 is replaced with Zinger's inpainting technique (i.e. the inpainting technique implemented in algorithm 3 step 6).

Soft depth comparison When the two textures are blended in algorithm 2 step 5, a hard depth comparison is performed to decide which pixels are rendered. The hard depth comparison results that the front-most pixel is rendered. We replace this with the soft-depth comparison used in algorithm 3 step 5. The soft depth comparison blends the pixels from the two views which have a small difference in depth, if the difference is large, only the front-most pixel is used.

Blur edges As a post-processing step, we use technique from Smolic et al. [42] to blur the edges (i.e. high depth discontinuities) in the rendered viewpoints. To do this the warped depth maps are blended and a Canny edge detector [44] is used on the blended depth to identify the edges. The edges are then dilated once so that they cover a 3 pixel wide border. We then copy the rendered viewpoint into a separate buffer and apply a 3x3 averaging low-pass filter over it, which blurs the image. The edge pixels in the blurred image is then extracted and copied into the rendered viewpoint giving us the final image. Blurring of edges is performed on all three base implementations.

5.5 Quality assessment

Subjective quality assessment of video is a thoroughly researched topic and remains the most robust method to evaluate the quality of video. Unfortunately conducting subjective evaluation tests of video is impractical since it requires a lot of time and money. Objective quality metrics allows for fast and inexpensive quality assessment and can also be used to monitor systems or optimize algorithms and parameters for the system.

An objective video quality metric can be used in three ways [45]:

- To monitor video quality for quality control systems.
- To benchmark image and video processing systems.
- To optimize algorithms and parameters.

Objective image and video quality metrics can be classified into three different categories according to the original image and video signal (which is considered distortion free and has perfect quality). We have full-reference (FR), reduced-reference (RR) and no-reference (NR).

- FR metrics compares the the distorted video to the distortion free reference video.
- RR metrics extracts features out of both videos and compares them.
- NR metrics have no reference video and is only based on the distorted video.

Peak signal-to-noise ratio (PSNR) and mean squared error (MSE) have been used extensively as objective quality metrics to assess video quality. These metrics have been widely adopted due to their clear physical meaning and simple calculation even if it has been show that they correlate poorly with the human visual system (HVS).

To remedy the situation, research has been put into finding objective quality criteria that better corresponds with the HVS. This research has lead to metrics such as structured similarity (SSIM), which better measure the perceived visual fidelity.

Although much research has gone into objective quality assessment of video, little research has been put into objective quality assessment of free-viewpoint rendering. Typically PSNR and MSE have been used for evaluating both depth and the quality of the rendered views. Even though objective quality metrics that have a stronger correlation to the human visual system have been proposed, they are designed to evaluate traditional transmission and compression systems, and have not been thoroughly tested to evaluate the synthesized views in free-viewpoint rendering. Video of interpolated views usually exhibit artefacts not typical in traditional video transmission/processing systems. It is therefore necessary to evaluate current objective quality approaches, identify common artefacts of free-viewpoint rendering and that the quality degradation that these artefacts introduce are properly measured by existing approaches.

A NR objective quality metric that detects ghosting artefacts in an interpolated image was proposed in [46]. Ghosting artefacts frequently occur in free-viewpoint rendering as depth estimation techniques are often inaccurate around depth discontinuities. Although this metric alone does not tell us much about the rendering quality.

A widely adopted technique to assess the quality of free-viewpoint rendering is to create a virtual viewpoint at the same position as a reference camera. A full-reference comparison can then be performed between the warped viewpoint and the image from the reference camera. The PSNR is frequently adopted as the metric for this comparison, and can be calculated by

$$PSNR = 10 \cdot log_{10}\left(\frac{MAX^2}{MSE}\right) \tag{5.5}$$

where MAX is the number of possible intensity values (usually 2^8) and MSE is the mean squared error. The MSE is calculated by

$$MSE = \frac{1}{w \cdot h} \sum_{u=1} \sum_{v=i} ||I_1(u,v) - I_2(u,v)||^2 \tag{5.6}$$

where w and h is the width and the height of the image and I_1 and I_2 are the pixel intensities at pixel coordinate (u, v). Since MSE and PSNR correlates poorly with the HVS, metrics like SSIM can also be used. The SSIM is calculated by

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \tag{5.7}$$

where μ_x, μ_y, σ_x, σ_y, σ_{xy} is the mean, variance and covariance of x and y, $c_1 = (k_1 L)^2$, $c_2 = (k_2 L)^2$ are two variables to stabilize the division with the weak denominator and L is the dynamic range of the pixel values. k_1 and k_2 should be set so that c_1 and c_2 only take effect when $(\mu_x^2 + \mu_y^2)$ or $(\sigma_x^2 + \sigma_y^2)$ is small.

The technique of warping to a virtual viewpoint and comparing the virtual image with the image from a reference camera can either be performed on synthetic data or on real data. The advantage of using synthetic data is that ground truth depth maps for the source cameras and precise calibration data can easily be obtained. With real images however, inaccuracies in calibration and depth estimation inevitably occurs. In addition noise and other distortions in the images can reduce the measured quality of the warped viewpoint compared to the reference camera. With synthetic images however, such inaccuracies and distortion does not occur, and only the performance of the rendering algorithm has an impact on the measured quality. Although quality assessment using synthetic image data does have some merit in that regard, its value is limited when the free-viewpoint pipeline is intended to be used on real data. In fact, the algorithms for free-viewpoint rendering takes into account inaccuracies of the estimated depth maps, such as ghosting artefact removal, which are not present when synthetic image data is used. In this thesis we therefore do the quality assessment on real data instead of synthetic data.

5.6 Experimental results

5.6.1 Experiment 1 - Base algorithms

To evaluate the rendering algorithms we use a commonly used procedure. We render a synthetic viewpoint at the same location as a reference camera. Under optimal conditions the rendered viewpoint and the viewpoint captured by the reference camera should be identical (however such an assumption would rarely true for any real rendering system). The image from the reference camera can then be regarded as a perfect signal with which we can compare to the rendered image, which is a full-reference evaluation. For the comparison of the images, we use two different metrics, PSNR and SSIM. In the evaluation we use images from the 'Ballet' and 'Breakdancers' data sets [17]. This data set was captured from 8 cameras configured in an arch. It includes 100 frames of each sequence, along with estimated calibration parameters and estimated depth maps. We use camera 2 and 4 as source cameras for the rendering, and synthesize an image at camera location 3 using the calibration data given for that camera. The captured image from camera 3 is then used as the perfect reference signal and we use all 100 frames from the data set and measure the PSNR and SSIM. The final results are then given as the arithmetic mean over all 100 frames.

Table 5.1: Measured PSNR and SSIM for the base implementations on the 'Ballet' and 'Breakdancers' sequences.

Ballet	PSNR	SSIM	Breakdancers	PSNR	SSIM
Mori	32.11	0.8560	Mori	32.98	0.7839
Morvan	32.10	0.8684	Morvan	33.02	0.8001
Zinger	32.10	0.8538	Zinger	32.90	0.7836

In table 5.1 we see the measured rendering quality of the three algorithms on the 'Ballet' and 'Breakdancers' data sets. The PSNR are roughly the same for all three algorithms, but algorithm 2 has somewhat higher SSIM. Subjectively however, algorithm 3 seemed to produce the best results, then algorithm 2 and the worst results were produced with algorithm 1. Examples of rendered viewpoints for the 'Ballet' and 'Breakdancers' sequences are shown in figure 5.14, 5.15 and 5.16.

Figure 5.14: Rendered viewpoints of the 'Ballet' and 'Breakdancers' sequence using algorithm 1.

Figure 5.15: Rendered viewpoints of the 'Ballet' and 'Breakdancers' sequence using algorithm 2.

Figure 5.16: Rendered viewpoints of the 'Ballet' and 'Breakdancers' sequence using algorithm 3.

5.6.2 Experiment 2 - Step-by-step evaluation of base algorithms

For the second experiment we applied the same method as we used in experiment 1. But instead of measuring PSNR and SSIM over the entire image, we measure the PSNR over a subset region of the image. This allows us to evaluate and compare the performance of the different steps in the rendering algorithms. For each step we measure PSNR over two different regions. The first region covers the pixels which we processed in that particular step, e.g. for the texture warping step, we only measure PSNR over the warped pixels and in the inpainting step we measure PSNR over the pixels which were inpainted. The second region we measure for each step, is the accumulated change, i.e. all the pixels which have been processed so far in the algorithm. E.g. when inverse warping the cracks in algorithm 3, PSNR is measured over the previously forward warped texture and the inverse warped cracks. These results can be seen in table 5.2, 5.3 and 5.4. The first column gives the PSNR over the processed pixels in that step, the second column gives the percentage of how many pixels were changed, the third column gives the accumulated PSNR over all changed pixels so far and the accumulated changed pixel percentage is given in the fourth column.

Algorithm 1 In table 5.2 we see the step-by-step quality measurement of algorithm 1. We see that the initial warping covers most of the image, 87.1% for 'Ballet' and 94.9% for 'Breakdancers'. Since the warping constitutes most of the image, the quality of the initial warping also has the most influence on the final PSNR. The dilation removes approximately 0.6 & - 0.7 % of the pixels. We see that the PSNR increases slightly after this step, which is expected since the dilation removes unreliable pixels around the disoccluded areas. We also see that the blending increases the PSNR significantly for both sequences, 0.80 dB for 'Ballet' and 0.92 dB for 'Breakdancers'. After the inpainting however, we get an approximately 0.3 dB decrease in PSNR for both sequences. This is caused by the very low PSNR over the inpainted regions, just 19.73 dB for 'Ballet' and 22.37 dB for 'Breakdancers'. The reason for this is that algorithm 1 use an inpainting method which does not rely on depth information, so background and foreground regions are smoothly interpolated. Also, there remains blended pixels around the remaining disoccluded regions after the blending (i.e. ghosting artefacts), which cause an additional deterioration of the quality. The overall PSNR is not greatly affected by

Table 5.2: Step-by-step quality measurement of algorithm 1 on the 'Ballet' and 'Breakdancers' sequences.

Ballet	PSNR, step	% pixels, step	PSNR, accumulated	% pixels, accumulated
Warping	31.52	87.1	31.52	87.1
Dilation	31.62	86.4	31.62	86.4
Blending	32.42	99.6	32.42	99.6
Inpainting	19.73	0.4	32.11	100.00

Breakdancers	PSNR, step	% pixels, step	PSNR, accumulated	% pixels, accumulated
Warping	32.23	94.9	32.23	94.9
Dilation	32.33	94.3	32.33	94.3
Blending	33.25	99.5	33.25	99.5
Inpainting	22.37	0.6	32.98	100.0

the poor inpainting however, as the inpainting is just done on 0.4%-0.6% of the pixels.

Algorithm 2 The initial warping of algorithm 2 achieves 0.60 dB - 0.72 dB increase in PSNR compared to algorithm 1. This increase occurs largely because algorithm 2 does bilinear interpolation when the textures are inverse warped, but also because the dilation/erosion operations on the warped depth maps increases the size of the disoccluded regions, which erase blended pixels. The gain in PSNR for blending is much less for algorithm 2 than algorithm 1, indicating that algorithm 2 has an inferior blending function.

Algorithm 3 Algorithm 3's warping has inferior PSNR to both previous algorithms, except for 'Ballet' which has higher PSNR than algorithm 1. Algorithm 3 omits warping textures located at high depth discontinuities, which can lead to background pixels being visible where foreground pixels should appear had the discontinuities not been excluded from the warp. The PSNR over the warped cracks is lower than the initially warped pixels. This is partly because the median filter fails to fill all the cracks in the warped depth map. Some of the cracks that appears in foreground objects are filled

Table 5.3: Step-by-step quality measurement of algorithm 2 on the 'Ballet' and 'Breakdancers' sequences.

Ballet	PSNR, step	% pixels, step	PSNR, accumulated	% pixels, accumulated
Warping	32.12	86.4	32.12	86.4
Blending	32.20	99.7	32.20	99.7
Inpainting	23.99	0.3	32.10	100

Breakdancers	PSNR, step	% pixels, step	PSNR, accumulated	% pixels, accumulated
Warping	32.95	94.2	32.95	94.2
Blending	33.25	99.5	33.25	99.5
Inpainting	22.12	0.5	33.02	100

with depth values from the background during the warping, and cause background texture to "shine through" the foreground object (see figure 5.17), thereby lowering the measured PSNR. For 'Ballet' the crack filling increase the PSNR by 0.89 dB, but for 'Breakdancers' there is barely no increase. By visual inspection, we observe that the shine-through occurs frequently in 'Ballet' but not in 'Breakdancers', the intensity difference between foreground and background is also larger in 'Ballet'. Another reason for the low PSNR is an artefact in the sample sequences. There is a 1-2 pixel wide line around the depth maps and textures with erroneous depth (i.e. it's very high) and color values. These lines are marked as cracks and constitute a relatively high percentage of the total number of pixels labeled as cracks, compared to the warping. Therefore they affect the measured PSNR in a larger degree for cracks than warping. Since these cracks have a very high depth value however, they are usually occluded after the blending, since algorithm 3 relies on a depth comparison. The depth based inpainting technique used for algorithm 3 has the highest measured PSNR of the three algorithms, but the regions to be inpainted is larger than the other algorithms, which can still lead to a lower overall PSNR.

Table 5.4: Step-by-step quality measurement of algorithm 3 on the 'Ballet' and 'Breakdancers' sequences.

Ballet	PSNR, step	% pixels, step	PSNR, accumulated	% pixels, accumulated
Warping	30.78	82.0	30.78	82.0
Cracks	30.13	5.7	31.67	85.7
Blending	32.24	99.2	32.24	99.2
Inpainting	25.39	0.8	32.10	100.0

Breakdancers	PSNR, step	% pixels, step	PSNR,step	% pixels, accumulated
Warping	32.35	90.4	32.35	90.4
Cracks	30.67	5.5	32.37	93.8
Blending	33.09	99.1	33.09	99.1
Inpainting	25.38	0.9	32.90	100

5.6.3 Artefacts and improvements of the base algorithms

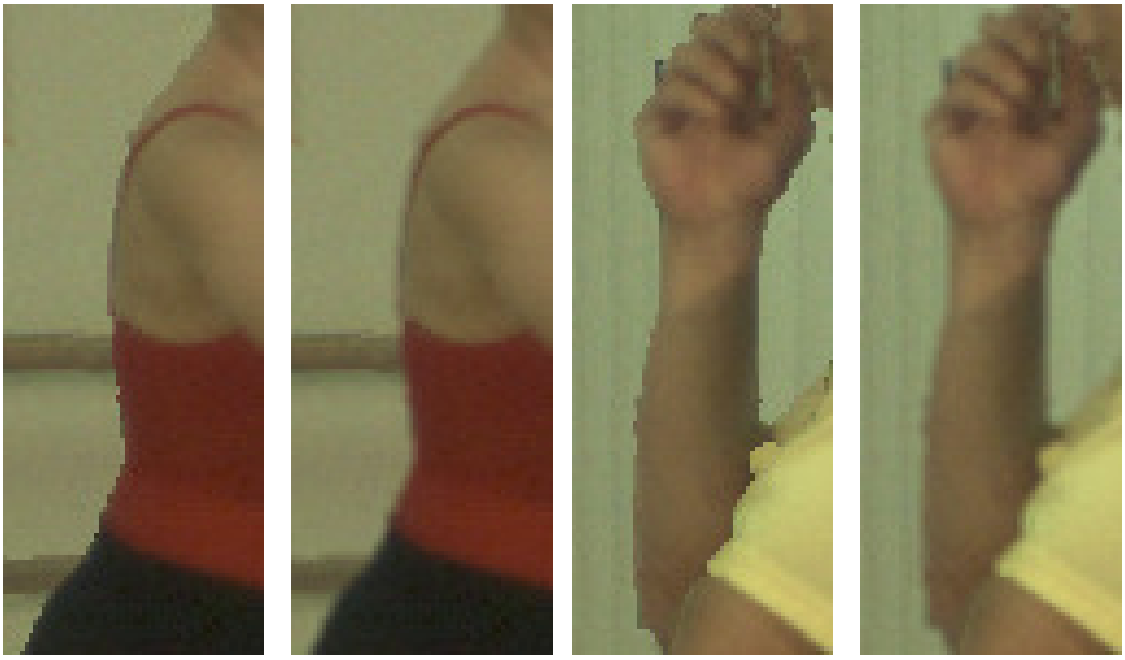
All three implemented warping algorithms, does in some form remove blended pixels that cause ghosting artefacts (or "inner ghosting"). The blended pixels cause a smooth transition between foreground and background objects. When these pixels are removed, the resulting warped edges can look sharp and unnatural, like the object was artificially inserted into the image, as shown in figure 5.18(a) and 5.18(c). To make it appear more natural, the edges in the texture is blurred. The results of this procedure is shown in figure 5.18(b) and 5.18(d), and clearly makes object borders appear more natural.

As previously explained, ghosting artefacts can appear at background regions after warping. However, blended pixels can also appear around the border of foreground objects. These artefacts are usually less visible since they appear on the border around objects, and it is not necessary to remove them. In fact, removing these pixels visibly decreases the size of the foreground objects and thus deteriorates the quality of the rendered view. When the blended pixels appear on object borders near disocclusions however, it can cause visually disturbing artefacts. In figure 5.19(a) and 5.19(c) we see

Figure 5.17: Shine-through artefacts in a warped viewpoint (before blending).

an example of "inner ghosting", when blended pixels appear not at the borders of an object, but inside the object. They appear inside the object because the disocclusions from one image is filled in by texture from another image, and the object borders from the images does not necessarily coincide, since boundary regions are often visible in one camera but occluded in the other.

The technique of not warping pixels at high depth discontinuities in algorithm 3 does not handle such artefacts well. If the blended pixels on foreground objects near high depth discontinuities are omitted from the warping, "inner ghosting" is removed, but it also significantly decreases the size of the object surface. Instead, dilating the disoccluded areas removes "inner ghosting" without removing blended pixels on foreground objects which are not around disoccluded regions, thus preserving the surface area of foreground objects. Although Zinger states that in dilating the disocclusions "..not only ghost contours will be removed, but also correct texture pixels." [39]. However, the good textures that are removed around disoccluded regions can often be re-



(a) Unnatural looking edges (b) Blurred edges (c) Unnatural looking edges (d) Blurred edges

Figure 5.18: In (a) and (c) we see hard and unnatural looking edges. In (b) and (b) the edges has been blurred using an averaging low-pass filter, blurring the edges and making them appear more natural.

trieved from the other texture. In figure 5.19(b) and 5.19(d) the disoccluded regions have been dilated, and indeed we see that the "inner ghosting" is removed.

Another problem with omitting high depth discontinuities from the warping is in the determination of edges. In algorithm 3's edge detection, a depth threshold is used to determine pixels at high discontinuities. The challenge is how to set the depth threshold properly. In figure 5.20 we see the results of edge detection using a low depth threshold, high depth threshold and no depth threshold (i.e. the disoccluded areas are instead dilated). In figure 5.20(a) we see a hole in the womans left arm, where the background shines through the foreground. This artefact occurred because the threshold was set so low that a high depth discontinuity was detected between the womans head and her left arm, and pixels at the "background" (i.e. at the arm) are omitted from the warping. Unfortunately, background pixels were warped to this location from one view, and in the other view the pixels were occluded. In figure 5.20(b) the depth thresh-

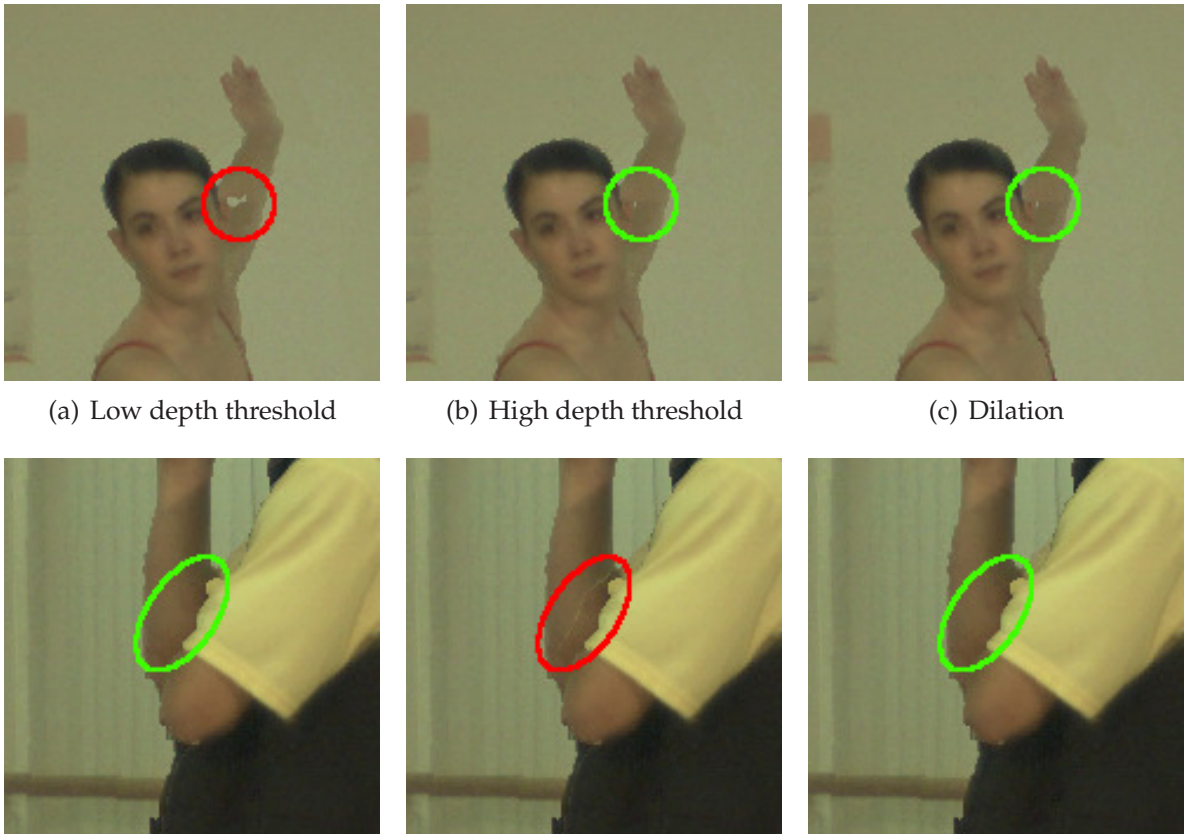
(a) Inner ghosting (b) Inner ghosting removed (c) Inner ghosting (d) Inner ghosting removed

Figure 5.19: In (a) and (c) we see inner ghosting artefacts on the foreground objects. These artefacts can be removed by dilating the disoccluded regions as shown in (b) and (d).

old has been set higher, so that less pixels are marked as edges, almost removing the artefact. However, increasing the depth threshold leads to another artefact as shown in figure 5.20(e). Here, blended pixels between the man's right arm and shirt are warped to the background (i.e. the arm), since it is no longer detected as an edge pixel. Using the lower threshold, as shown in figure 5.20(d), gave no such artefacts. In this case, we can not set the depth threshold so that both the artefacts are removed. By dilating the disoccluded regions however, as shown in figure 5.20(c) and 5.20(f), both of these artefacts are removed.

5.6.4 Experiment 3 - Variations

To deal with the artefacts mentioned in the previous section and to increase the quality of the rendered view, we tested alternative implementations of the three algorithms. We tested different approaches to remove blended pixels, the effects of bilinear interpolation, better inpainting techniques and different crack filling techniques (i.e. post-processing of the warped depth maps) to see what effects it had on objective and sub-



(a) Low depth threshold (b) High depth threshold (c) Dilation

(d) Low depth threshold (e) High depth threshold (f) Dilation

Figure 5.20: In (a) and (d), the edges threshold has been set low, which cause a hole to appear in the womans arm. In (b) and (e) the threshold is set higher, which removes the hole artefacts but instead causes a new artefact to appear, where pixels from the shirt are warped to the man's arm. In (c) and (f) no threshold is used, but instead relies on dilating the disoccluded regions. The size of the womans arm is reduced and no artefacts are visible on the man's arm.

jective visual quality.

Algorithm 1

Table 5.5: Measured PSNR and SSIM difference using variations on algorithm 1 on the 'Ballet' and 'Breakdancers' sequences.

Algorithm 1 variations	Ballet		Breakdancers	
	PSNR	SSIM	PSNR	SSIM
Vanilla	32.11	0.8560	32.98	0.7839
No bilateral	+0.01	-0.0001	0.00	-0.0002
Bilinear interpolation	+0.29	+0.0177	+0.28	+0.0194
Blur edges	+0.32	+0.0031	+0.13	+0.0025
2px dilation	-0.16	-0.0005	-0.16	-0.0008
2px dilation w/cross-check	-0.13	-0.0009	-0.11	-0.0012
w/Zinger inpainting	+0.27	+0.0002	+0.11	+0.0006

No bilateral filtering We skipped the bilateral filtering of the warped depth maps for algorithm 1. As we can see from table 5.5 the measured PSNR and SSIM difference from not doing bilateral filtering is very small. Subjectively, the rendered viewpoints are virtually indistinguishable. The viewpoints were slightly difference, but no quality gain could be observed using bilateral filtering.

Bilinear interpolation For the inverse warping of the textures we used bilinear interpolation as in algorithm 2. This lead to a 0.3dB gain in PSNR and 0.0177-0.0194 increase in SSIM. Visually, we saw a clear improvement in the 'Ballet' sequence, where the bilinear interpolation resulted in anti-aliased rendering of the colored vertical lines on the wall as shown in figure 5.21. The quality improvement for the 'Breakdancers' sequence was not so prominent, but for both sequences we also observed that borders around foreground objects were not so sharp, leading to a more natural look.

Blur edges The blurring of edges gave a 0.13 dB-0.32 dB PSNR increase and 0.0025-0.0031 SSIM increase. Also the subjective quality was improved, in that foreground

(a) No bilinear interpolation (b) Bilinear interpolation

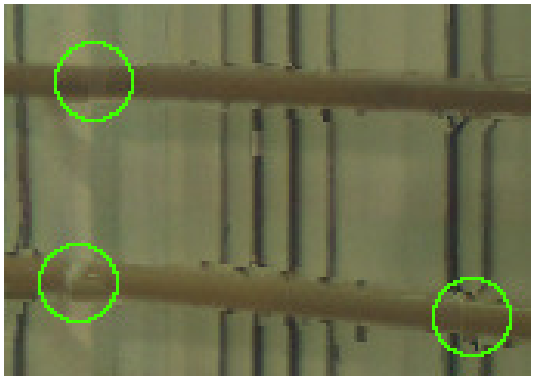
Figure 5.21: In (a) no bilinear interpolation is done and cause aliasing of the vertical lines. The viewpoint in (b) bilinear interpolation is done, giving anti-aliased rendering of the vertical lines.

objects appeared more natural as shown in figure 5.18.

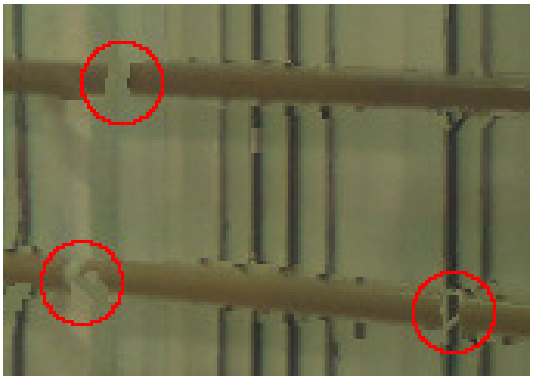
Disocclusion dilation Although both the alternative disocclusion dilation techniques gave a reduction in PSNR and SSIM, they remove visually disturbing artefacts. The amount of ghosting artefacts were significantly reduced using either of these methods, including the appearance of inner ghosting. Using 2 px dilation also significantly improved the measured PSNR over the inpainted regions, even if this approach gave the lowest PSNR over the entire image. The dilation with cross-checking removed some artefacts, but the pure 2px dilation gave superior visual result, even if the measured objective quality was lower.

Zinger inpainting Zinger inpainting increased the measured quality, however some artefacts are still created from the inpainting. This is because of blended pixels (i.e. ghosting artefacts) that still remain at the background regions. In some regions, this inpainting technique performs worse, e.g. at the bars of the wall in the 'Ballet' sequence as shown in figure 5.22. Since the bars have a lower depth value than the wall, the

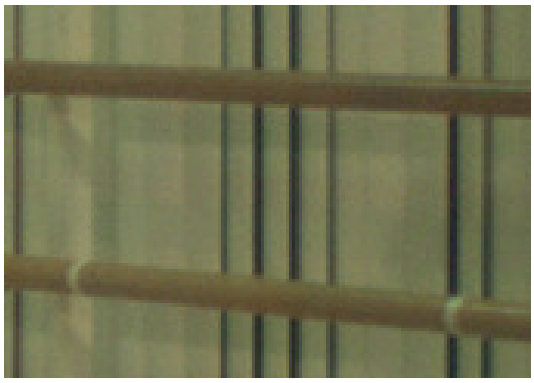
region is incorrectly inpainted with texture from the wall. The overall visual quality is still better, since it does not do a smooth interpolation between background and foreground regions.



(a) Vanilla inpainting for algorithm 2



(b) Zinger inpainting for algorithm 2



(c) Reference camera

Figure 5.22: Different inpainting techniques in algorithm 2. In (a) the vanilla inpainting is performed. In (b) Zinger inpainting is done, causing artefacts in the rendered viewpoint. For comparison the image from the reference camera is shown in (c).

Algorithm 2

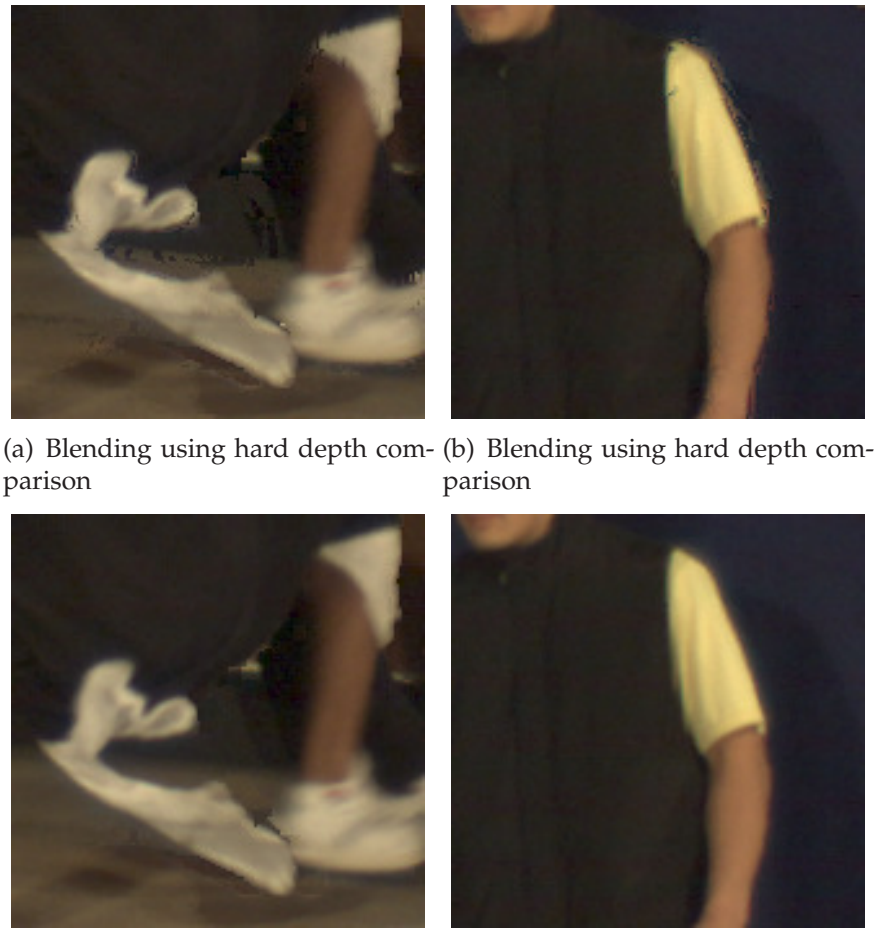
Disocclusion dilation The dilation/erosion of the warped depth maps in algorithm 2 expands the disoccluded regions, but some disturbing ghosting artefacts still remain. Both alternative dilation methods remove inner and outer ghosting artefacts, but it does not have such a large change on the PSNR as in algorithm 1. This is likely due to the better inpainting technique used for algorithm 2.

Table 5.6: Measured PSNR and SSIM difference using variations on algorithm 2 on the 'Ballet' and 'Breakdancers' sequences.

Algorithm 2 variations	Ballet		Breakdancers	
	PSNR	SSIM	PSNR	SSIM
Vanilla	32.10	0.8684	33.02	0.8001
1px dilation	-0.02	+0.0008	-0.07	-0.0002
1px dilation w/cross-check	+0.01	+0.0008	+0.00	+0.0000
blur edges	+0.16	+0.0021	+0.13	+0.0021
soft depth compare	+0.23	+0.0043	+0.21	+0.0027
w/Zinger inpainting	+0.05	+0.0000	+0.04	+0.0004

Blur edges The gain in PSNR and SSIM was smaller for algorithm 2 than algorithm 1. This is due to the bilinear interpolation, which also cause edge regions to be smoothed somewhat. Still, the perceived quality is improved as foreground objects does not appear like they have been artificially inserted into the scene.

Soft depth comparison Algorithm 2's blending function is based on color and depth information. If neither pixel is disoccluded and the colors are similar, they are blended. If the colors are dissimilar, a hard depth comparison is done so that the front-most pixel is used. However, we have observed that colors can be dissimilar, even if the have approximately the same depth. Also, depth information in general is not reliable enough to just use the front-most pixel. Using such a hard depth compare can lead to artefacts as shown in figure 5.23(a) and 5.23(b). Estimated depth on texture-less regions can be unreliable, and when two viewpoints are warped, textures might not align exactly at these areas. Using color similarity does not always give good results. In algorithm 1, the entire textures are blended (i.e. no depth comparison), although this can lead to blending of foreground and background objects. In algorithm 3 we use a soft depth comparison, such that if the absolute difference of the depth pixels are under a small threshold (i.e. the depths are close to each other), they are blended. This gives a much more visually pleasing result as shown in figure 5.23(c) and 5.23(d) and a 0.21 - 0.23 dB increase in PSNR and 0.0027-0.0043 increase in SSIM.



(a) Blending using hard depth comparison

(b) Blending using hard depth comparison

(c) Blending using soft depth comparison

(d) Blending using soft depth comparison

Figure 5.23: In (a) and (b), the textures are blended using a hard depth comparison where the front-most pixels is rendered, leading to visually disturbing artefacts. In (c) and (d) a soft depth comparison is used instead, where depth values that lie close to each other cause a blending of pixels, leading to less artefacts and a more visually pleasing look.

Zinger inpainting Although the PSNR and SSIM gain for using Zinger inpainting were quite small, we could see a significant visual improvement, especially on the 'Ballet' sequence. When disoccluded regions are covered by foreground texture on both sides, the vanilla inpainting technique for algorithm 2 falls short in comparison to the Zinger inpainting approach. For the 'Breakdancers' sequence however, the it was not clear which method gave the best visual result.

Algorithm 3

Table 5.7: Measured PSNR and SSIM difference using variations on algorithm 3 on the 'Ballet' and 'Breakdancers' sequences.

Algorithm 3 variations	Ballet		Breakdancers	
	PSNR	SSIM	PSNR	SSIM
Vanilla	32.10	0.8538	32.90	0.7836
5x5 median	+0.04	+0.0007	+0.02	+0.0005
1+1 px edge/dilate	-0.13	-0.0011	-0.12	-0.0001
1+1 px edge/dilate w/5x5 median	-0.03	+0.0007	-0.03	+0.0004
1+1 px edge/dilate w/cross-check	-0.02	-0.0002	-0.02	-0.0007
1+1 px edge/dilate w/cross-check and 5x5 median	+0.01	+0.0011	+0.01	+0.0004
2px dilate	-0.24	-0.0025	-0.26	-0.0025
2px dilate w/5x5 median	-0.07	+0.0004	-0.11	-0.0002
2px dilate w/cross-check	-0.09	-0.0013	-0.06	-0.0021
2px dilate w/cross-check and 5x5 median	-0.03	+0.0008	-0.03	-0.0003
Bilinear interpolation	+0.32	+0.0178	+0.30	+0.0196
Blur edges	+0.32	+0.0048	+0.18	+0.0033

5x5 median filtering The 3x3 median filter on the warped depth maps does not fill all the small cracks in the image. This sometimes results in background pixels shining through foreground objects. This artefact however, is usually removed when the images are blended, since a depth comparison is performed, which cause the foreground pixel to be rendered instead of blending it with the background, and since these pixels usually does not appear in the same location in both of the warped textures. When they do, the use of a 5x5 median filter helps filling in these pixels and reduce the size

of large holes that shine through the foreground. These artefacts are much easier to spot in the 'Ballet' sequence, since the intensity difference between the foreground and background is much larger than for the 'Breakdancers' sequence. The measured quality improvement is small, and it is only visible in a handful of frames.

1 pixel edge detection + 1 pixel disocclusion dilation All the alternative 1 pixel edge detection and 1 pixel dilation performed better than the vanilla implementation at removing inner ghosting artefacts. They also reduced the size of shine through holes, and using the 5x5 median filter improved the results further. The dilation sometimes reduced the surface area of foreground objects, This was especially visible when it occurred around thin foreground objects, such as fingers and arms. Using the cross-checking constraint reduced the amount of good pixels removed. The variations not using cross-checking or the 5x5 median filter suffered from some pixelation artefacts. The changes were most visible in the 'Ballet' sequence, whereas in the 'Breakdancers' sequence, it was often not obvious if the quality was improved or degraded. All the variations except 1 pixel edge detection + 1 pixel disocclusion dilation with 5x5 median filter and cross-checking gave a decrease in PSNR, although using the larger median filter gave a slight increase in SSIM.

2 pixel disocclusion dilation The 2 pixel dilation also performs better when it comes to removing inner ghosting and reducing the size of shine-through holes. However, not using cross-checking introduced pixelation artefacts in the image and removed parts of foreground objects. Also, the removal of good foreground pixels is much more apparent than for 1 pixel edge detection + 1 pixel dilation. Cross-checking improved this, but instead introduce disturbing inpainting artefacts. Again, the changes were less visible in the 'Breakdancers' sequence, except for the reduction in foreground object size. There is a slight increase in SSIM for the two dilation methods using the 5x5 median filter on the 'Ballet' sequences, all the other variations give a decrease in PSNR and SSIM. Also, the objective quality is lower than the 1 pixel edge detection + 1 pixel dilation methods.

Bilinear interpolation The bilinear interpolation gave similar results as observed in the corresponding algorithm 1 variation.

Blur edges The edges rendered using algorithm 3 were sharper and more unnatural looking than that of algorithm 1, which also resulted in a slightly higher SSIM increase. Otherwise the results were similar to that of the corresponding algorithm 1 variation.

Chapter 6

Discussion, conclusion and future work

6.1 Discussion

6.1.1 Calibration

We calibrated our cameras by using readily available functions in the OpenCV library. The implementation gives fairly good results. However, there are some variations in the calibration data, indicating that we do not get the true parameters of the system. There are several reasons for these variations. Calibration by using a printed planar pattern is inherently not accurate enough for finding the true parameters of the cameras. We have numerous sources of error, e.g. the printer does not print the points with sufficient precision, the checkerboard should have been put on a flatter surface with proper adhesion, so that it would be more planar. We taped our checkerboard onto a white-board, although it would probably be better to glue the pattern onto the surface to get it more flat.

6.1.2 Depth estimation

Out of the five tested disparity estimation algorithms, three of them (SGBM, BP and CSBP) had reasonably good results with 10.4 % to 12.3 % bad pixels on average on the four different data sets . The remaining two algorithms (BM, and BM on GPU) got significantly worse results ranging from 28.4 % to 41.6 % bad pixels on average. Visually it was also apparent that these two algorithms produced disparity maps with a signif-

icant amount of holes in them, which is detrimental to free-viewpoint rendering. Also SGBM suffered from holes in the disparity map, although not to the extent as the BM algorithms. It is likely that the quality of the disparity maps could be significantly increased by applying a hole-filling algorithm, e.g. by extrapolating background pixels. However, because of the sheer number and large size of the holes in the BM algorithms, filling the holes would probably not lead to a quality comparable to SGBM, BP or CSBP. SGBM has far less holes in it and the holes appear mostly near discontinuous depth regions. It is likely that many of these holes appear because they are occluded in one of the cameras, and that they are in fact background regions. Therefore, extrapolating background pixels in these holes could lead to an increase in the quality of the estimated disparity map and lead to better view synthesis. Although BP and CSBP gave good results on the test data, the results on our captured data (i.e. 'Man') did not yield such good results. 'Man' exhibits quite different properties than the other sequences. It has much larger disparities and more texture-less regions. By setting the parameters for BP and CSBP carefully, better quality could be achieved. By visual inspections it appears as SGBM also yields good disparity estimates also on 'Man', although this could not be verified by objective metrics since no ground truth disparities are available.

Only the BM algorithms and SGBM were able to achieve real-time performance (i.e. frame-rate more than 10 fps). BP just achieved 1 fps and CSBP even less at 0.15 fps. SGBM appears as a good candidate for a free-viewpoint video application, as it is robust and estimates accurate disparity with a high frame-rate. Since it is performed on the CPU, it also leaves the GPU free to perform other task, such as rendering. Performance tests were done on a low-end GPU, a more powerful one could increase the frame-rate of BP and CSBP, but the large memory requirements for these algorithms could still be an issue, especially if the resolution of the input images are increased (naturally this would also lead to a decrease in frame-rate).

6.1.3 Rendering

The three base implementations of free-viewpoint rendering scored similar PSNR and SSIM, although the SSIM of algorithm 2 was slightly better. However, there was some differences in visual quality in the three implementations. However, a small number of pixels can create disturbing visual artefacts without having a large impact on PSNR.

In addition to the base implementations, we tested a large number of variations of the algorithms. Several of these variations lead to a decrease in visual artefacts and increase in PSNR and SSIM. However, subjective visual quality and measured objective quality was not always consistent with each other. PSNR and SSIM are valuable tools to assess the quality of the rendered viewpoints, but not necessarily decisive. This is a challenge, since many published free-viewpoint rendering algorithms report PSNR to validate the quality of their method. Although PSNR can be a good indicator of quality, it can sometimes be unreliable, if not misleading. Optimizing a free-viewpoint algorithm on PSNR alone, without considering the subjective quality would be inadvisable.

6.2 Conclusion

In this thesis we investigated how to achieve a high-quality free-viewpoint pipeline. We implemented and examined the three essential components of such a pipeline, namely camera calibration, depth estimation and free-viewpoint rendering. We took an experimental approach and tested several depth estimation and rendering algorithms in terms of quality and performance.

We showed that calibration robustness can be increased by taking a large number of images of the calibration pattern.

Three of the five disparity estimation algorithms (SGBM, BP and CSBP) achieved high-quality disparity estimates. However, only SGBM gives high-quality disparity maps and with real-time performance.

We implemented three different free-viewpoint rendering algorithms adopted from literature. They all gave similar results in the measured quality using PSNR and SSIM, although there were some subjective quality differences visible in the different output views. We further tested a large number of variations of the rendering algorithms, demonstrating that quality could be further improved. In addition, we presented a novel constraint for removing "ghosting" artefacts in the rendered viewpoints using cross-checking. The cross-checking constraint ensures that pixels around disoccluded regions in one warped viewpoint are not removed unless they are visible in the other viewpoint.

6.3 Future work

After examining the quality of the calibration, we found that it does give sufficient accuracy for high-quality free-viewpoint rendering, although the calibration can be further improved by using a high-quality 3D calibration pattern.

Although SGBM gave good results, some holes remained in the estimated disparity map. For free-viewpoint rendering, all pixels should be assigned a depth value. This method can therefore be extended by filling in the holes, e.g. by extrapolation nearby background pixels.

Our implemented rendering algorithms focused on achieving high-quality view synthesis. These algorithms must be optimized for real-time performance, e.g. by hardware acceleration.

Bibliography

[1] Shenchang Eric Chen. Quicktime vr: an image-based approach to virtual environment navigation. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '95, pages 29–38, New York, NY, USA, 1995. ACM.

[2] Richard Szeliski and Heung-Yeung Shum. Creating full view panoramic image mosaics and environment maps. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '97, pages 251–258, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[3] Leonard McMillan and Gary Bishop. Plenoptic modeling: an image-based rendering system. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '95, pages 39–46, New York, NY, USA, 1995. ACM.

[4] Marc Levoy and Pat Hanrahan. Light field rendering. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '96, pages 31–42, New York, NY, USA, 1996. ACM.

[5] Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. The lumigraph. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '96, pages 43–54, New York, NY, USA, 1996. ACM.

[6] Shenchang Eric Chen and Lance Williams. View interpolation for image synthesis. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '93, pages 279–288, New York, NY, USA, 1993. ACM.

[7] Steven M. Seitz and Charles R. Dyer. View morphing. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '96, pages 21–30, New York, NY, USA, 1996. ACM.

[8] L. McMillan Jr. *An image-based approach to three-dimensional computer graphics*. PhD thesis, University of North Carolina at Chapel Hill, 1997.

[9] Jonathan Shade, Steven Gortler, Li-wei He, and Richard Szeliski. Layered depth images. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '98, pages 231–242, New York, NY, USA, 1998. ACM.

[10] Paul E. Debevec, Camillo J. Taylor, and Jitendra Malik. Modeling and rendering architecture from photographs: a hybrid geometry- and image-based approach. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '96, pages 11–20, New York, NY, USA, 1996. ACM.

[11] Takeo Kanade, Peter Rander, and P. J. Narayanan. Virtualized Reality: Constructing Virtual Worlds from Real Scenes. *IEEE MultiMedia*, 4:34–47, January 1997.

[12] Saied Moezzi, Arun Katkere, Don Y. Kuramura, and Ramesh Jain. Reality modeling and visualization from multiple video sequences. *IEEE Comput. Graph. Appl.*, 16:58–63, November 1996.

[13] Wojciech Matusik, Chris Buehler, Ramesh Raskar, Steven J. Gortler, and Leonard McMillan. Image-based visual hulls. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '00, pages 369–374, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[14] B. Milburn, M. Smulski, H. H. K. Lee, and H Horowitz. The light field video camera. *SPIE Electronic Imaging; Media Processors*, 4674:29–36, January 2002.

[15] V. Vaish, B. Wilburn, N. Joshi, and M. Levoy. Using plane + parallax for calibrating dense camera arrays. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2004.

[16] Joel Carranza, Christian Theobalt, Marcus A. Magnor, and Hans-Peter Seidel. Free-viewpoint video of human actors. *ACM Trans. Graph.*, 22:569–577, July 2003.

[17] C. Lawrence Zitnick, Sing Bing Kang, Matthew Uyttendaele, Simon Winder, and Richard Szeliski. High-quality video view interpolation using a layered representation. *ACM Trans. Graph.*, 23:600–608, August 2004.

[18] G.R. Bradski and A. Kaehler. *Learning OpenCV*. O'Reilly, 2008.

[19] A. Albarelli, E. Rodolà, and A. Torsello. Robust Camera Calibration using Inaccurate Targets. *Trans. Pattern Anal. Mach. Intell*, 31(2):376–383, 2009.

[20] L. Lucchese and SK Mitra. Using saddle points for subpixel feature detection in camera calibration targets. In *Circuits and Systems, 2002. APCCAS'02. 2002 Asia-Pacific Conference on*, volume 2, pages 191–195. IEEE, 2002.

[21] J. Mallon and P.F. Whelan. Which pattern? Biasing aspects of planar calibration patterns and detection methods. *Pattern recognition letters*, 28(8):921–930, 2007.

[22] Z. Zhang. A flexible new technique for camera calibration. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(11):1330–1334, 2000.

[23] D.C. Brown. Close-range camera calibration. *Photogrammetric engineering*, 37(8):855–866, 1971.

[24] F. Remondino and C. Fraser. Digital camera calibration methods: considerations and comparisons. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 36(5):266–272, 2006.

[25] Derek Bradley and Wolfgang Heidrich. Binocular camera calibration using rectification error. In *Proceedings of the 2010 Canadian Conference on Computer and Robot Vision*, CRV '10, pages 183–190, Washington, DC, USA, 2010. IEEE Computer Society.

[26] A. Ruiz and G. Garcia-Mateos. A note on principal point estimability. In *Pattern Recognition, 2002. Proceedings. 16th International Conference on*, volume 2, pages 304–307. IEEE, 2002.

[27] Daniel Scharstein and Richard Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *Int. J. Comput. Vision*, 47:7–42, April 2002.

[28] C. Nvidia. Compute unified device architecture programming guide. *NVIDIA: Santa Clara, CA*, 83:129, 2007.

[29] K. Konolige. Small vision systems: Hardware and implementation. In *ROBOTICS RESEARCH-INTERNATIONAL SYMPOSIUM-*, volume 8, pages 203–212. MIT PRESS, 1998.

[30] H. Hirschmüller. Stereo processing by semiglobal matching and mutual information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 328–341, 2008.

[31] P.F. Felzenszwalb and D.P. Huttenlocher. Efficient belief propagation for early vision. *International journal of computer vision*, 70(1):41–54, 2006.

[32] Q. Yang, L. Wang, and N. Ahuja. A constant-space belief propagation algorithm for stereo matching. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2010.

[33] Steven M. Seitz, Brian Curless, James Diebel, Daniel Scharstein, and Richard Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 1*, pages 519–528, Washington, DC, USA, 2006. IEEE Computer Society.

[34] D. Scharstein and R. Szeliski. High-accuracy stereo depth maps using structured light. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2003.

[35] A. Telea. An image inpainting technique based on the fast marching method. *JOURNAL OF GRAPHICS TOOLS.*, 9(1):23–34, 2004.

[36] Y.Y. Morvan. *Acquisition, compression and rendering of depth and texture for multi-view video*. PhD thesis, Technische Universiteit Eindhoven, 2009.

[37] Yuji Mori, Norishige Fukushima, Tomohiro Yendo, Toshiaki Fujii, and Masayuki Tanimoto. View generation with 3d warping using depth information for ftv. *Signal Processing: Image Communication*, 24(1-2):65 – 72, 2009. Special issue on advances in three-dimensional television and video.

[38] K.J. Oh, S. Yea, and Y.S. Ho. Hole filling method using depth based in-painting for view synthesis in free viewpoint television and 3-d video. In *Picture Coding Symposium, 2009. PCS 2009*, pages 1–4. IEEE, 2009.

[39] S. Zinger, L. Do, et al. Free-viewpoint depth image based rendering. *Journal of Visual Communication and Image Representation*, 21(5-6):533–541, 2010.

[40] William R. Mark, Leonard McMillan, and Gary Bishop. Post-rendering 3d warping. In *Proceedings of the 1997 symposium on Interactive 3D graphics*, I3D '97, pages 7–ff., New York, NY, USA, 1997. ACM.

[41] M. Schmeing and X. Jiang. Depth image based rendering: A faithful approach for the disocclusion problem. In *3DTV-Conference: The True Vision-Capture, Transmission and Display of 3D Video (3DTV-CON), 2010*, pages 1–4. IEEE.

[42] A. Smolic, K. Muller, K. Dix, P. Merkle, P. Kauff, and T. Wiegand. Intermediate view interpolation based on multiview video plus depth for advanced 3D video systems. In *Image Processing, 2008. ICIP 2008. 15th IEEE International Conference on*, pages 2448–2451. IEEE, 2008.

[43] L. Do, S. Zinger, et al. Quality improving techniques for free-viewpoint dibr. In *Proceedings of SPIE, the International Society for Optical Engineering*. Society of Photo-Optical Instrumentation Engineers, 2010.

[44] J. Canny. A computational approach to edge detection. *Readings in computer vision: issues, problems, principles, and paradigms*, 184(87-116):86, 1987.

[45] Z. Wang, H.R. Sheikh, and A.C. Bovik. Objective video quality assessment. *The Handbook of Video Databases: Design and Applications*, pages 1041–1078, 2003.

[46] Kai Berger, Christian Lipski, Christian Linz, Anita Sellent, and Marcus Magnor. A ghosting artifact detector for interpolated image quality assessment. In *Proceedings of the 6th Symposium on Applied Perception in Graphics and Visualization*, APGV '09, pages 128–128, New York, NY, USA, 2009. ACM.